RESEARCH PROJECT

TITLE: Conditional Shortest Path Routing in Delay Tolerrant Networks

1. INTRODUCTION

1.1 Introduction To Project

This article studies Delay tolerant networks (DTN's) where each node knows the probabilistic distribution of contacts with other nodes. Delay tolerant networks are characterized by the sporadic connectivity between their nodes and therefore the lack of stable end-to-end paths from source to destination. Since the future node connections are mostly unknown in these networks, opportunistic forwarding is used to deliver messages. Based on the observations about human mobility traces and the findings of previous work, we introduce a new metric called conditional intermeeting time. We propose Conditional Shortest Path Routing (CSPR) protocol that route the messages over conditional shortest paths in which the cost of links between nodes is defined by conditional intermeeting times rather than the conventional intermeeting times. When a node receives a message from one of its contacts, it stores the message in its buffer and carries the message until it encounters another node which is at least as useful (in terms of the delivery) as itself. Through trace-driven simulations, we demonstrate that CSPR achieves higher delivery rate and lower end-to-end delay compared to the shortest path based routing protocols that use the conventional intermeeting time as the link metric.

1.2 Problems In Existing System

A connected hypercube with faulty links and/or nodes is called an injured hypercube. A distributed adaptive fault-tolerant routing scheme is proposed for an injured hypercube in which each node is required to know only the condition of its own links. Despite its simplicity, this scheme is shown to be capable of routing messages successfully in an injured n-dimensional hypercube as long as the number of faulty components is less than n. Moreover, it is proved that this scheme routes messages via shortest paths with a rather high probability, and the expected length of a resulting path is very close so that of a shortest path. Since the assumption that the number of faulty components is less than n in an n-dimensional hypercube might limit the usefulness of the above scheme, a routing scheme based on depth-first search which works in the presence of an arbitrary number of faulty components is introduced. Due to the insufficient information on faulty components, however, the paths chosen by this scheme may not always be the shortest. To guarantee all messages to be routed via shortest paths, the authors propose to equip

every node with more information than that on its own links. The effects of this additional information on routing efficiency are analyzed, and the additional information to be kept at each node for the shortest path routing is determined.

1.3 Solution To These Problems

In this proposed system, we redefine the intermeeting time concept between nodes and introduce a new link metric called conditional intermeeting time. It is the intermeeting time between two nodes given that one of the nodes has previously met a certain other node. This updated definition of intermeeting time is also more convenient for the context of message routing because the messages are received from a node and given to another node on the way towards the destination. Here, conditional intermeeting time represent the period over which the node holds the message. To show the benefits of the proposed metric, we propose conditional shortest path routing (CSPR) protocol in which average conditional intermeeting times are used as link costs rather than standard intermeeting times and the messages are routed over conditional shortest paths (CSP). We compare CSPR protocol with the existing shortest path (SP) based routing protocol through real trace- driven simulations. The results demonstrate that CSPR achieves higher delivery rate and lower end-to-end delay compared to the shortest path based routing protocols. This show how well the conditional intermeeting time represents internode's link costs (in the context of routing) and helps making effective forwarding decisions while routing a message. Routing algorithms in DTN's utilize a paradigm called store-carry-and-forward. We generated the multiple messages from a random source node to a random destination node at each t seconds. Clearly, CSPR algorithm delivers more messages than SPR algorithm.

2. LITERATURE SURVEY

An analysis of real mobility traces has been done in different environments (office , conference , city , skating tour) with different objects (human, bus, zebra) and with variable number of attendants and led to significant results about the aggregate and pair wise mobility characteristics of real objects. Recent analysis on real mobility traces have demonstrated that models assuming the exponential distribution of intermeeting times between pairs of nodes do not match real data well. Instead up to 99% of intermeeting times in many datasets is log-normal distribution. This makes the pair wise contacts between nodes depend on their pasts. Such a finding invalidates a common assumption that the pair wise intermeeting times are exponentially distributed and memory less. More formally, if X is the random variable representing the intermeeting time between two nodes, $P(X > s + t \mid X > t)$ _= $P(X > s)$ for s, t > 0. Hence, the residual time until the next meeting of two nodes can be predicted better if the node knows that it has not met the other node for t time units.

Algorithm 1 update (node m, time t)

1: **if** m is seen first time **then**
2: firstTimeAt[m] $\leftarrow t$
3: **else**
4: increment β_m by 1
5: lastTimeAt[m] $\leftarrow t$
6: **end if**
7: **for** each neighbor $j \in N$ and $j \neq m$ **do**
8: start a timer t_{mj}
9: **end for**
10: **for** each neighbor $j \in N$ and $j \neq m$ **do**
11: **for** each timer t_{jm} running **do**
12: $S[j][m]$ += time on t_{jm}
13: increment $C[j][m]$ by 1
14: **end for**
15: delete all timers t_{jm}
16: **end for**
17: **for** each neighbor $i \in N$ **do**
18: **for** each neighbor $j \in N$ and $j \neq i$ **do**
19: **if** $S[j][i] \neq 0$ **then**
20: $\tau_s(i|j) \leftarrow S[j][i] \ / \ C[j][i]$
21: **end if**
22: **end for**
23: $\tau_s(i) \leftarrow$ (lastTimeAt[i] $-$ firstTimeAt[i]) / β_i
24: **end for**

In Algorithm 1, each node first adds up times expired between repeating meetings of one neighbor and the meeting of another neighbor. Then it divides this total by the number of times it has met the first neighbor prior to meeting the second one. For example, if node *A* has two neighbors (*B* and *C*), to find the conditional intermeeting time of $\tau A(B|C)$, each time node *A* meets node *C*, it starts a different timer. When it meets node *B*, it sums up the values of these timers and divides the results by the number of active timers before deleting them. This computation is repeated again each time node *B* is encountered. Then, the total of times collected from each timer is divided by the total count of timers used, to find the value of $\tau A(B|C)$.

Network Model:

We model a DTN as a graph *G* = (*V, E*) where the vertices (*V)* are mobile nodes and the edges (*E*) represent the connections between these nodes. However, different from previous DTN network models, we assume that there may be multiple unidirectional (*Eu*) and bidirectional (*Eb*) edges between the nodes. The neighbors of a node *i* are denoted with *N* (*i*) and the edge sets are given as follows:

$$\begin{aligned} E &= E_u \cup E_b \\ E_b &= \{(i,j) \mid \forall j \in N(i)\} \text{ where, } w(i,j) = \tau_i(j) = \tau_j(i) \\ E_u &= \{(i,j) \mid \forall j,k \in N(i) \text{ and } j \neq k\} \text{ where,} \\ &\quad w(i,j) = \tau_i(j|k) \end{aligned}$$

The above definition of *Eu* allows for multiple unidirectional edges between any two nodes. However, these edges differ from each other in terms of their weights and the corresponding third node. This third node indicates the previous meeting and is used as a reference point while defining the conditional intermeeting time (weight of the edge). We illustrate a sample DTN graph with four nodes and nine edges. Of these nine edges, three are bidirectional with weights of standard intermeeting times between nodes, and six are unidirectional edges with weights of conditional intermeeting times.

Conditional Shortest Path Routing:

Our algorithm basically finds conditional shortest paths (CSP) for each source-destination pair and routes the messages over these paths. We define the CSP from a node *n*0 to a node *nd* as follows:

$$CSP(n_0, n_d) = \{n_0, n_1, \ldots, n_{d-1}, n_d \mid \Re_{n_0}(n_1|t) + \sum_{i=1}^{d-1} \tau_{n_i}(n_{i+1}|n_{i-1}) \text{ is minimized.}\}$$

Here, t represents the time that has passed since the last meeting of node $n0$ with $n1$ and $\Re_{n_0}(n_1|t)$ is the expected residual time for node $n0$ to meet with node $n1$ given that they have not met in the last t time units. $\Re_{n_0}(n_1|t)$ can be computed with parameters of distribution representing the intermeeting time between $n0$ and $n1$. It can also be computed in a discrete manner from the contact history of $n0$ and $n1$.

Assume that node i observed d intermeeting times with node j in its past. Let τ 1

$$\Re_i(j|t) = \frac{\sum_{k=1}^{d} f_i^k(j)}{|\{\tau_i^k(j) \geq t\}|} \text{ where,}$$
$$f_i^k(j) = \begin{cases} \tau_i^k(j) - t & \text{if } \tau_i^k(j) \geq t \\ 0 & \text{otherwise} \end{cases}$$

Here, if none of the d observed intermeeting times is bigger than t (this case occurs less likely as the contact history grows), a good approximation can be to assume $\Re_i(j|t) = 0$. We will next provide an example to show the benefit of CSP over SP. The weights of edges (A, C) and (A, B) show the expected residual time of node A with nodes C and B respectively in both graphs. But the weights of edges (C, D) and (B, D) are different in both graphs. While in the left graph, they show the average intermeeting times of nodes C and B with D respectively, in the right graph, they show the average conditional intermeeting times of the same nodes with D relative to their meeting with node A. From the left graph, we conclude that SP(A, D) follows (A,B,D). Hence, it is expected that on average a message from node A will be delivered to node D in 40 time units.

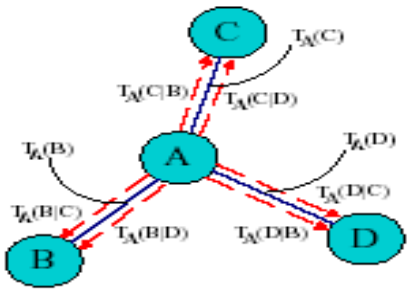


Fig-1. The graph of a sample DTN with four nodes and nine edges in total.

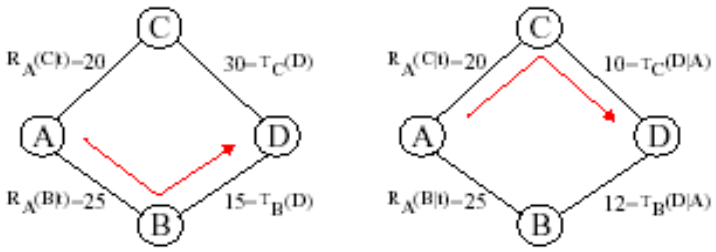


Fig-2. The shortest path from a source to destination node can be different when conditional intermeeting times are used as the weights of links in the network graph.

However this may not be the actual shortest delay path. As the weight of edge (*C*, *D*) states in the right graph, node *C* can have a smaller conditional intermeeting time (than the standard intermeeting time) with node *D* assuming that it has met node *A*. This provides node *C* with a faster transfer of the message to node *D* after meeting node *A*. Hence, in the right graph, CSP(*A*, *D*) is (*A,C,D*) with the path cost of 30 time units.

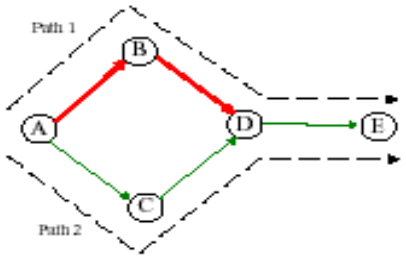


Fig- 3. Path 2 may have smaller conditional delay than path 1 even though the CSP from node A to node D is through node B.

Each node forms the aforementioned network model and collects the standard and conditional intermeeting times of their nodes between each other through epidemic link state protocol. However, once the weights are known, it is not as easy to find CSP's as it is to find SP's. Consider Figure 5 where the CSP(*A*, *E*) follows path 2 and CSP(*A*, *D*) follows (*A*, *B*, *D*). This situation is likely to happen in a DTN,

if $\tau D(E|B) \geq \tau D(E|C)$ is satisfied. Running Dijkstra's or Bellman-ford algorithm on the current graph structure cannot detect such cases and concludes that CSP(*A*, *E*) is (*A*, *B*, *D*, *E*). Therefore, to obtain the correct CSP's for each source destination pair, we propose the following transformation on the current graph structure.

Given a DTN graph $G = (V,E)$, we obtain a new graph $G' = (V', E')$ where:

$$
\begin{aligned}
V' &\subseteq V \times V \text{ and } E' \subseteq V' \times V' \text{ where,} \\
V' &= \{(i_j) \mid \forall j \in N(i)\} \text{ and } E' = \{(i_j, k_l) \mid i = l\} \\
&\text{where, } w'(i_j, k_l) = \begin{cases} \tau_i(k|j) & \text{if } j \neq k \\ \tau_i(k) & \text{otherwise} \end{cases}
\end{aligned}
$$

Note that the edges in *Eb* (in *G*) are made directional in *G'* and the edges in *Eu* between the same pair of nodes are separated in *E'*. This graph transformation keeps all the historical information that conditional intermeeting times require and also keeps only the paths with a valid history. For example, for a path *A,B,C,D* in *G*, an edge like (*CD,DA*) in *G'* cannot be chosen because of the edge settings in the graph. Hence, only the correct τ values will be added to the path calculation. To solve the CSP problem however, we add one vertex for source *S* (apart from its permutations) and one vertex for destination node *D*. We also add outgoing edges from *S* to each vertex $(i_S) \in V'$ with weight $\Re_S(i|t)$. Furthermore, for the destination node, *D*, we add only incoming edges from each vertex $i_j \in V'$ with weight $\tau i(D|j)$.

Routing in a Delay Tolerant Network

1. Introduction

In this work, we look at the problem of routing in a delay tolerant network (DTN)[8]. Such networks are assumed to experience frequent, long-duration partitioning and may never have an end-to-end contemporaneous path. This problem contrasts with routing in conventional data networks which typically selects a shortest policy-compliant path in a connected graph without considering availability of intermediate buffering and bandwidth capacity.

In graph theoretic terms, our problem is a form of the \quickest transshipment problem" in which both edge capacities and transit delays along an edge can vary (down to zero) as a function of time and nodes have finite buffers [12]. In practical terms, DTNs arise in networks with known connectivity patterns such as Low-Earth Orbiting Satellites (LEO) or those with unpredicted, opportunistic connectivity (e.g., communication among PDAs when brought into close proximity [5]).Here, we focus on the former case.

The routing problem in a DTN may at first appear as the standard problem of dynamic routing but with extended link failure times. This is not the case. For the standard dynamic routing problem, the topology is assumed to be connected (or partitioned for very short intervals), and the objective of the routing algorithm is to find the best currently-available path to move traffic end-to-end. In a DTN, however, an end-to- end path may be unavailable at all times; routing is performed over time to achieve eventual delivery by employing long-term storage at the intermediate nodes. The DTN routing problem amounts to a constrained optimization problem where edges may be unavailable for extended periods of time and a storage constraint exists at each node. This formulation reveals DTN routing to be a considerably different and more challenging problem.

In this paper, we make several contributions: we first motivate and formulate the DTN routing problem when the connectivity patterns are known, then provide a framework for evaluating various routing algorithms, and finally show a simulation-based comparison of several of our own algorithms. We also include an optimal algorithm based on a linear programming approach to serve as a basis for comparison with the simulations. Finally, we outline the future work to be accomplished in the area.

2. Example: Connecting a Remote Village

The problem of providing data communications to remote and rural areas is beginning to attract the attention of the computer systems research community [23]. While many rural connectivity projects involve attempts to provide conventional Internet access to remote areas, a small number of projects are taking an alternative approach which focuses on asynchronous messaging in order to greatly reduce the cost of connectivity [25, 20, 23]. For example, the Wizzy Digital Courier service provides asynchronous (disconnected) Internet access to schools in remote villages of South Africa [25]. In this system, a courier on a motorbike, equipped with a USB storage device, travels from a village school to a large city which has permanent (reasonably high-speed) Internet connectivity. Typically, it takes a few hours for the courier to travel from the village to the city.

In consideration of this scenario, we realize that several other connectivity options may be available (e.g. satellites, either LEO or GEO, possibly telephone), but are not likely to be cost effective or of sufficient capacity to handle all of the traffic. Conversely, for some traffic, such as high-priority alerts, low latency may be sufficiently important to justify using a higher- cost communication system offering smaller delay. Thus, we consider a simple extended scenario, based on this real-world example that motivates the DTN routing problem.

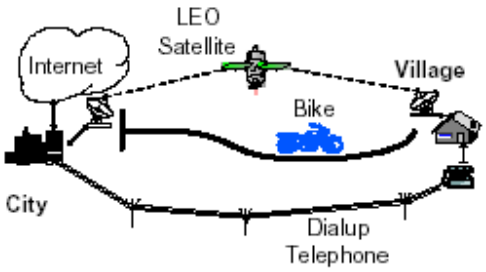


Fig-4. Scenario illustrating a variety of connectivity options between a remote village and a city. Even in this simple scenario, many route choices are possible.

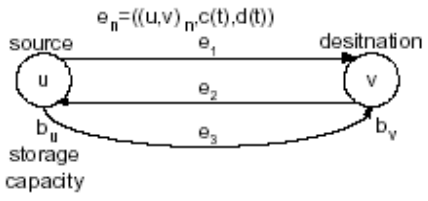
Figure 1 shows a hypothetical village served by a digital courier, a wired dialup Internet connection, and a store-and- forward LEO satellite (e.g. PACSAT). These satellites have low to moderate bandwidth (around 10 Kbps) and are visible for 4-5 short periods of time (\passes") per day (lasting around 10 minutes per pass, depending on the orbit inclination and location on Earth). We call the opportunity to communicate a contact (as in [8]), which is characterized by a duration

of time, a capacity, and a propagation delay (assumed to remain constant during the contact duration). In addition, depending on the type of connection used, buffering constraints may also need to be considered.1 The digital courier service represents a high-bandwidth, high-latency contact, the dialup represents a low-bandwidth, low-latency contact, and the LEO satellite represents a moderate-bandwidth, moderate-latency contact. The problem of selecting which contacts to carry messages and when represents an instance of the DTN routing problem. Route selection may depend on a variety of factors including message source and destination, size, time of request, available contacts, traffic in the system, or other factors (e.g. cost, delay, etc.).

In the next sections we develop a set of definitions and a framework for evaluating DTN routing algorithms. We then propose several of our own routing algorithms and use the framework in conjunction with simulations to evaluate their performance in the context of this village scenario.

3. DTN NETWORK MODEL

Nodes and Edges The DTN graph is a directed multi-graph, in which more than one edge (also called link) may exist between a pair of nodes (see Figure 2). The reason for using a multi graph is straightforward: it may be possible to select between two distinct (physical) connection types to move data between the same pair of nodes. Furthermore, the link capacities (and to a lesser extent, propagation delay) are time-dependent (capacity is zero at times when the link is unavailable). Thus, the set of edges in the graph must capture both time varying capacity and propagation delay as well as multiple parallel edges. A simple example of an edge captured by this description involves a ground station and a LEO satellite rising, passing directly overhead, and setting at the opposite horizon.



Edges in a DTN graph. Nodes may be connected by multiple edges, representing different physical links. Each node j performs store-and-forward routing, and has finite storage capacity (bj). An edge is parameterized by its source and destination nodes plus a capacity (c(t)) and delay function (d(t)).

As it rises, its channel capacity will generally increase until it is directly overhead and will decrease for the remaining time of the pass. This is because noise is minimal when the satellite is directly overhead but increases at lower elevations. Another example would be a bus (carrying a wireless access point) passing by a village. The throughput of the wireless link would depend upon the distance of the bus from the village. When no communication is possible, the edge is assigned zero capacity.

Contact

A contact is an opportunity to send data over an edge. More precisely, it is a specific edge and a corresponding time interval during which the edge capacity is strictly positive.

Messages

Communication demands are represented by messages. A message is a tuple (u; v; t;m), where u is the source of the message, v is the destination, t is the time at which the message is injected into the system and m is its size (messages can be of arbitrary size). The set of all messages is called the traffic demand.

Storage

The nodes in a DTN have finite long-term storage (buffers) used for holding in-transit data or data waiting to be consumed by the application at a destination node. In our model, the storage is exclusively used for holding in-transit data. Destination nodes are assumed to have sufficient capacity for holding data to be consumed by an application.

Routing

Routing occurs in a store and forward fashion. The routing algorithm is responsible for determining the next edge(s) that a message should be forwarded along. Messages not immediately forwarded wait until they are assigned to contacts by the routing algorithm.

4. DTN Routing Issues

In this section, we consider a number of important issues in any routing algorithm: the routing objective, the amount of knowledge about the network required by the scheme, when

routes are computed, the use of multiple paths, and the use of source routing. We focus on how these issues arise in the context of the DTN routing problem.

4.1 Routing Objective

The routing objective of traditional routing schemes has been to select a path which minimizes some simple metric (e.g. the number of hops). For DTN networks, however, the most desirable objective is not immediately obvious. One natural objective is to maximize the probability of message delivery. Messages could potentially be lost due to creation of a routing loop or the forced discarding of data when buffers are exhausted. As an approximation, we focus on minimizing the delay of a message (the time between when it is injected and when it is completely received).

While DTN applications are expected to be tolerant of delay, this does not mean that they would not benefit from decreased delay. Furthermore, we believe this metric is an appropriate measure to use in exploring the differential evaluation of several routing algorithms in an application-independent manner. Minimizing delay lowers the time messages spend in the network, reducing contention for resources (in a qualitative sense). Therefore, lowering delay indirectly improves the probability of message delivery. This is validated by our simulation results.

4.2 Proactive Routing vs. Reactive Routing

In proactive routing, routes are computed automatically and independently of traffic arrivals. Most Internet standard routing protocols and some ad-hoc protocols such as DSDV (Destination Sequenced Distance Vector) and OLSR (Optimized Link-State Routing) are examples of this style [4]. In a DTN, these protocols are capable of computing routes for a connected sub graph of the overall DTN topology graph. They fail when asked to provide paths to nodes which are not currently reachable. Despite this drawback, proactive network-layer routing protocols may provide useful input to DTN routing algorithm by providing the set of currently-reachable nodes from which DTN routing may select preferred next hops. In reactive routing, routes are discovered on-demand when traffic must be delivered to an unknown destination. Ad-hoc routing protocols such as AODV (Ad-hoc On-demand Distance Vector) and DSR (Dynamic Source Routing) are examples of this style [4]. In these systems, a route discovery protocol is employed to determine routes to destinations on-demand, incurring additional delay. These protocols work best when communication patterns are relatively sparse. For a DTN, as with the proactive protocols, these

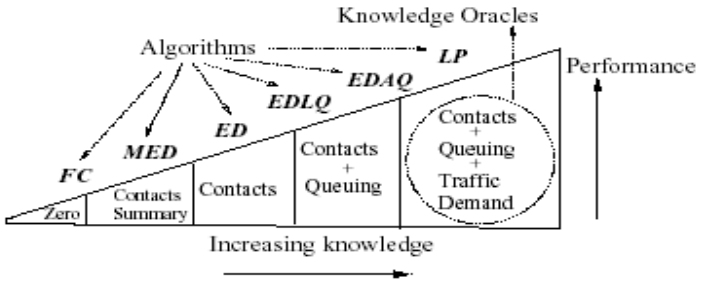
protocols work only for finding routes in a connected sub graph of the overall DTN routing graph. However, they fail in a different way than the proactive protocols. In particular, they will simply fail to return a successful route (from a lack of response), whereas the proactive protocols can potentially fail more quickly (by determining that the requested destination is not presently reachable).

In a DTN, routes may vary with time in predictable ways and can be pre-computed using knowledge about future topology dynamics. Employing a proactive approach would likely involve computing several sets of routes and indexing them by time. The associated resource requirements would be prohibitive unless the traffic demand is large and a large percentage of the possible network nodes exchange traffic. Otherwise, a reactive approach would be more attractive.

A related issue is route stability, a measure of how long the currently-known routes are valid. Route stability depends on the rate of topological change. With relatively stable routes one can employ route caching to avoid unnecessary routing protocol exchanges. With future knowledge about topology changes, caching could be especially effective in a DTN because it may be possible to know ahead of time exactly when to evict existing cached route entries.

4.3 Source Routing vs Perhop Routing

In source routing the complete path of a message is determined at the source node, and encoded in some way in the message. The route is therefore determined once and does not change as the message traverses the network. In contrast, in per-hop routing the next-hop of a message is determined at each hop along its forwarding path. Per-hop routing allows a message to utilize local information about available contacts and queues at each hop, which is typically unavailable at the source. Thus, per-hop routing may lead to better performance. Unfortunately, due to its local nature, it may lead to loops when nodes have different topological views (e.g. due to incomplete or delayed routing information).



Conceptual performance vs knowledge trade-off. The x-axis depicts the amount of knowledge (increasing in the positive direction). The y-axis depicts the expected performance that can be achieved using a certain amount of knowledge. The figure shows that more knowledge is required to attain better performance. Labels on top show algorithms developed in this paper using the corresponding oracles.

4.4 Message Splitting

A message is split when forwarded in such a way that different parts (fragments) are routed along different paths (or across different contacts on the same path). This technique may reduce the delay or improve load balancing among multiple links. It is particularly relevant in DTNs because messages can be arbitrarily large and may not _t in a single contact. However, splitting complicates routing because, in addition to determining the sizes of the fragments, we also have to determine corresponding paths for the fragments.

4.5 Computing Shortest (minimum cost) Paths

To model the forwarding delay of a message in a DTN, we consider three delay components: 1) queuing time: time until a contact becomes available, 2) transmission delay: time to inject a message completely into an edge, and 3) propagation delay. Queuing time includes both the time waiting for an edge to become available (waiting time) plus the time to drain messages already scheduled for departure on that edge. Queuing time can be large because edges may be unavailable for long periods of time. Given that edge capacities and propagation delays vary with time, we also expect route selection to vary with time. When edge costs are time-invariant, shortest paths can be computed using Dijkstra's shortest path algorithm. However, if the costs are changing with time the straightforward approach does not work. We must make two modifications to overcome this problem. First, the time a message will arrive at a particular node must be predicted. Second, the predicted arrival time must be used to determine the cost of taking subsequent edges. This would, in turn, affect the time the message arrives at neighboring nodes. Interestingly,

Dijkstra's algorithm can be adapted to compute the shortest paths for this case. Pseudo-code for the modified algorithm is given in Algorithm. The key difference between this algorithm and the traditional Dijkstra's algorithm is the definition and the use of the w (cost) function. It takes into account the time a message arrives at a node. This time is then used to compute the cost of traversing edges emanating from that node (lines 7 and 8 of Algorithm).

The modified Dijkstra's algorithm requires the cost function for all edges to have the FIFO property. This property ensures that a message can not arrive earlier at the destination of an edge by simply waiting longer at the source of the edge (i.e. you will not travel more quickly over an edge if you wait to use it). Formally, it means that for all edges e and all pairs of time t1; t2 with t1 < t2, w(t1; e) + t1 _ w(t2; e) + t2. When considering different cost assignments, we make sure the above condition holds. In practice it does because, in a single physical link, any message would not be able overtake other messages sent earlier.

Abbr.	Name	Description	Oracles Used
FC	First Contact	Use any available contact	None
MED	Minimum Expected Delay	Dijkstra with time-invariant edge costs based on average edge waiting time	Contacts Summary
ED	Earliest Delivery	Modified Dijkstra with time-varying cost function based on waiting time	Contacts
EDLQ	Earliest Delivery with Local Queue	ED with cost function incorporating local queuing	Contacts
EDAQ	Earliest Delivery with All Queue	ED with cost function incorporating queuing information at all nodes and using reservations	Contacts and Queuing
LP	Linear Program	-	Contacts, Queuing and Traffic

Table 1: Overview of different routing algorithms. All Dijkstra-based algorithms incorporate a cost function sensitive to edge propagation and transmission delays. Costs are ascertained by consulting the respective oracles.

```
Input: G = (V,E), s, T, w(e,t)
Output: L
 1: Q ← {V}
 2: L[s] ← 0 , L[v] ← ∞ ∀ v ∈ V s.t v ≠ s.
 3: while Q ≠ { } do
 4:    Let u ∈ Q be the node s.t L[u] = min_{x∈Q} L[x]
 5:    Q = Q ← {u}
 6:    for each edge e ∈ E, s.t. e = (u,v)  do
 7:       if L[v] > (L[u] + w(e, L[u] + T)) then
 8:          L[v] ← L[u] + w(e, L[u] + T)
 9:       end if
10:    end for
11: end while
```

Algorithm 1: Dijkstra's Algorithm modified to use time- varying edge costs. s is the source node. T is the start time. L : V ! R is the array returning the cost of the shortest path for all nodes. The cost function w : E _ R+ ! R+, gives the cost as a function of edge and time. The interpretation of w(e; t) is the following: Let e be an edge from node u to node v. Given a message at u at time t, w(e; t) is the cost (delay) of sending it to v. Therefore, if e is taken the message will reach v at time t+w(e; t). The algorithm also works if the network topology is a multi graph. The unmodified Dijkstra's algorithm (for time invariant costs) is the same except that the cost function w(e; L[u] + T) is replaced by w(e) in lines 7 and 8.

The above property does not prevent a message from waiting at a node in order to reduce the overall delay. For example, consider a case in which two nodes have two edges between them with different propagation delays. The algorithm may prefer to wait for the lower propagation delay link over the other even if it is currently unavailable.

3. FEASIBILITY REPORT

Preliminary investigation examine project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

- Technical Feasibility
- Operational Feasibility
- Economical Feasibility

3.1 Technical Feasibility

The technical issue usually raised during the feasibility stage of the investigation includes the following:

- Does the necessary technology exist to do what is suggested?
- Do the proposed equipments have the technical capacity to hold the data required to use the new system?
- Will the proposed system provide adequate response to inquiries, regardless of the number or location of users?
- Can the system be upgraded if developed?
- Are there technical guarantees of accuracy, reliability, ease of access and data security?

Earlier no system existed to cater to the needs of 'Secure Infrastructure Implementation System'. The current system developed is technically feasible. It is a web based user interface for audit workflow at NIC-CSD. Thus it provides an easy access to the users. The database's purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security. The software and hard requirements for the development of this project are not many and are already available in-house at NIC or are available as free as open source. The work for the project is done with the current equipment and existing software technology. Necessary bandwidth

exists for providing a fast feedback to the users irrespective of the number of users using the system.

3.2 Operational Feasibility

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization's operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. Some of the important issues raised are to test the operational feasibility of a project includes the following:

- Is there sufficient support for the management from the users?
- Will the system be used and work properly if it is being developed and implemented?
- Will there be any resistance from the user that will undermine the possible application benefits?

This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits.

The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

3.3. Economical Feasibility

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, There is nominal expenditure and economical feasibility for certain.

4. SYSTEM ANALYSIS

After analyzing the requirements of the task to be performed, the next step is to analyze the problem and understand its context. The first activity in the phase is studying the existing system and other is to understand the requirements and domain of the new system. Both the activities are equally important, but the first activity serves as a basis of giving the functional specifications and then successful design of the proposed system. Understanding the properties and requirements of a new system is more difficult and requires creative thinking and understanding of existing running system is also difficult, improper understanding of present system can lead diversion from solution.

4.1 SDLC Methodologies

This document play a vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

Spiral Model

It was defined by Barry Boehm in his 1988 article, "A spiral Model of Software Development and Enhancement. This model was not the first model to discuss iterative development, but it was the first model to explain why the iteration models.

As originally envisioned, the iterations were typically 6 months to 2 years long. Each phase starts with a design goal and ends with a client reviewing the progress thus far. Analysis and engineering efforts are applied at each phase of the project, with an eye toward the end goal of the project.

The steps for Spiral Model can be generalized as follows:

- The new system requirements are defined in as much details as possible. This usually involves interviewing a number of users representing all the external or internal users and other aspects of the existing system.
- A preliminary design is created for the new system.

The following diagram shows how a spiral model acts like:

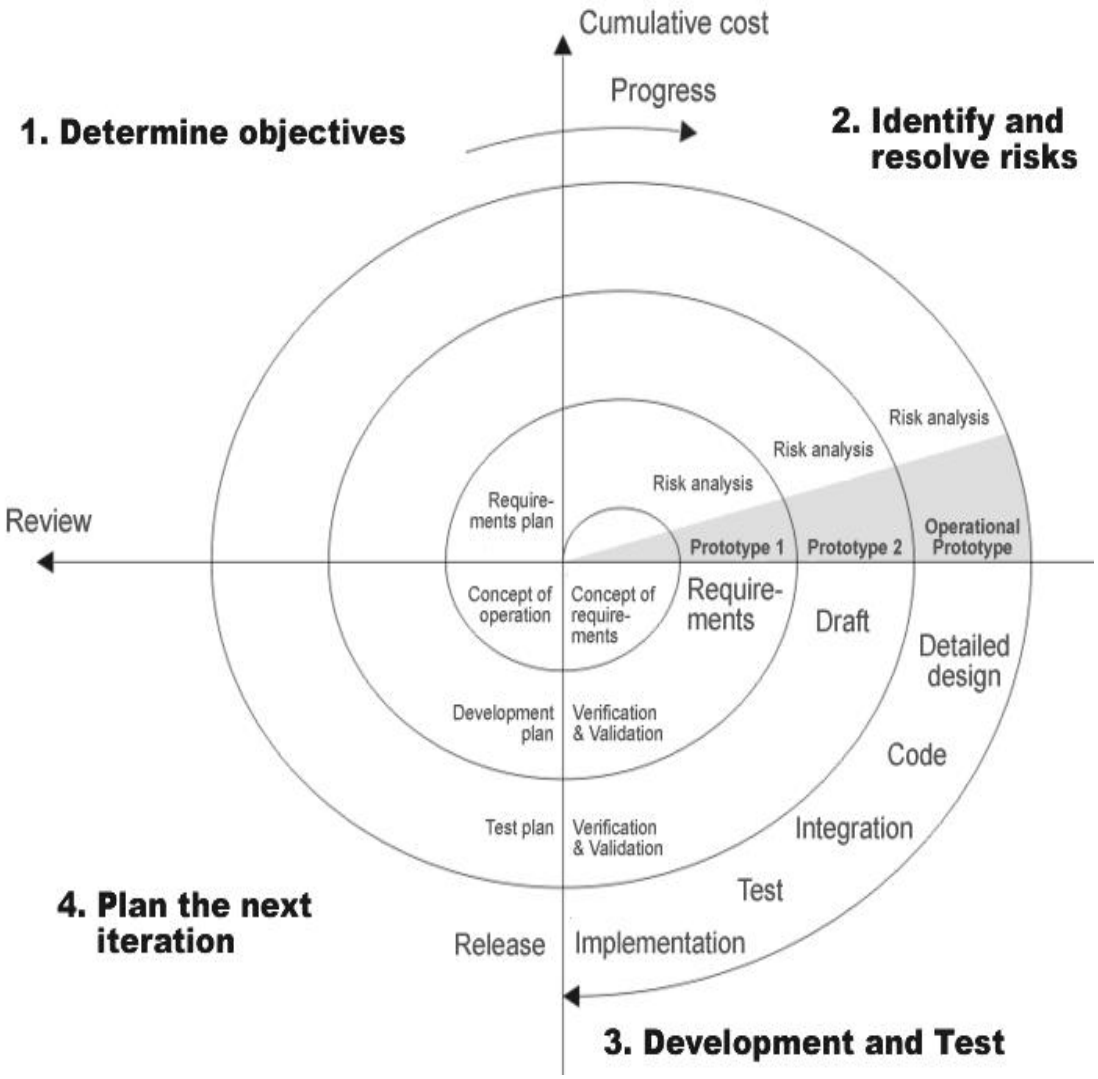


Fig-5 -Spiral Model

- A first prototype of the new system is constructed from the preliminary design. This is usually a scaled-down system, and represents an approximation of the characteristics of the final product.
- A second prototype is evolved by a fourfold procedure:
    1. Evaluating the first prototype in terms of its strengths, weakness, and risks.
    2. Defining the requirements of the second prototype.

3. Planning an designing the second prototype.

4. Constructing and testing the second prototype.

- At the customer option, the entire project can be aborted if the risk is deemed too great. Risk factors might involved development cost overruns, operating-cost miscalculation, or any other factor that could, in the customer's judgment, result in a less-than-satisfactory final product.
- The existing prototype is evaluated in the same manner as was the previous prototype, and if necessary, another prototype is developed from it according to the fourfold procedure outlined above.
- The preceding steps are iterated until the customer is satisfied that the refined prototype represents the final product desired.
- The final system is constructed, based on the refined prototype.
- The final system is thoroughly evaluated and tested. Routine maintenance is carried on a continuing basis to prevent large scale failures and to minimize down time.

4.2 Application Development

N-Tier Application

N-Tier Applications can easily implement the concepts of Distributed Application Design and Architecture. The N-Tier Applications provide strategic benefits to Enterprise Solutions. While 2-tier, client-server can help us create quick and easy solutions and may be used for Rapid Prototyping, they can easily become maintenance and security might mare. The N-tier Applications provide specific advantages that are vital to the business continuity of the enterprise.

Typical features of a real life n-tier may include the following:

- Security
- Availability and Scalability
- Manageability
- Easy Maintenance

- Data Abstraction

The above mentioned points are some of the key design goals of a successful n-tier application that intends to provide a good Business Solution.

Definition:

Simply stated, an n-tier application helps us distribute the overall functionality into various tiers or layers:

- Presentation Layer
- Business Rules Layer
- Data Access Layer
- Database/Data Store

Each layer can be developed independently of the other provided that it adheres to the standards and communicates with the other layers as per the specifications.

This is the one of the biggest advantages of the n-tier application. Each layer can potentially treat the other layer as a 'Block-Box'.

In other words, each layer does not care how other layer processes the data as long as it sends the right data in a correct format.

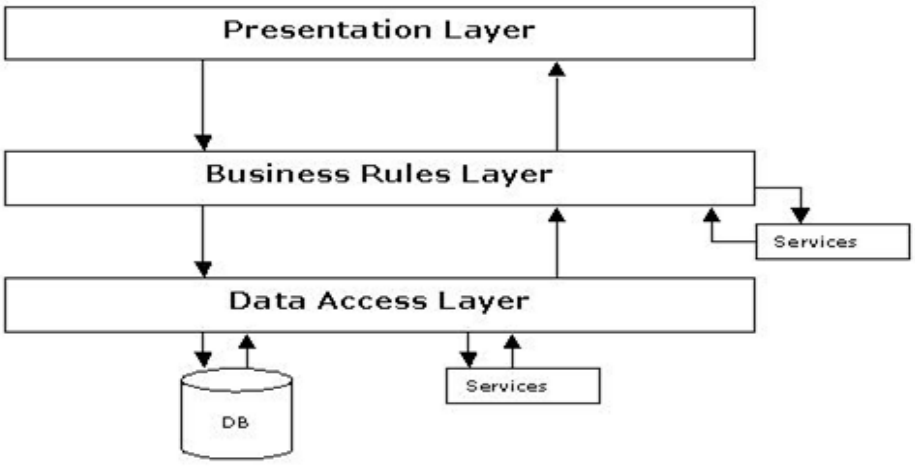


Fig-6 -N-Tier Architecture

1. The Presentation Layer

Also called as the client layer comprises of components that are dedicated to presenting the data to the user.

For example: Windows/Web Forms and buttons, edit boxes, Text boxes, labels, grids, etc.

2. The Business Rules Layer

This layer encapsulates the Business rules or the business logic of the encapsulations. To have a separate layer for business logic is of a great advantage. This is because any changes in Business Rules can be easily handled in this layer. As long as the interface between the layers remains the same, any changes to the unctionality/processing logic in this layer can be made without impacting the others. A lot of client-server apps failed to implement successfully as changing the business logic was a painful process.

3. The Data Access Layer

This layer comprises of components that help in accessing the Database. If used in the right way, this layer provides a level of abstraction for the database structures. Simply put changes made to the database, tables, etc do not affect the rest of the application because of the Data Access layer. The different application layers send the data requests to this layer and receive the response from this layer.

4. The Database Layer

This layer comprises of the Database Components such as DB Files, Tables, Views, etc. The Actual database could be created using SQL Server, Oracle, Flat files, etc.
In an n-tier application, the entire application can be implemented in such a way that it is independent of the actual Database. For instance, you could change the Database Location with minimal changes to Data Access Layer. The rest of the Application should remain unaffected.

4.3 Proposed System

In this proposed system, we redefine the intermeeting time concept between nodes and introduce a new link metric called conditional intermeeting time. It is the intermeeting time between two nodes given that one of the nodes has previously met a certain other node. This

updated definition of intermeeting time is also more convenient for the context of message routing because the messages are received from a node and given to another node on the way towards the destination. Here, conditional intermeeting time represent the period over which the node holds the message. To show the benefits of the proposed metric, we propose conditional shortest path routing (CSPR) protocol in which average conditional intermeeting times are used as link costs rather than standard intermeeting times and the messages are routed over conditional shortest paths (CSP). We compare CSPR protocol with the existing shortest path (SP) based routing protocol through real trace- driven simulations. The results demonstrate that CSPR achieves higher delivery rate and lower end-to-end delay compared to the shortest path based routing protocols. This show how well the conditional intermeeting time represents internode's link costs (in the context of routing) and helps making effective forwarding decisions while routing a message. Routing algorithms in DTN's utilize a paradigm called store-carry-and-forward. We generated the multiple messages from a random source node to a random destination node at each t seconds. Clearly, CSPR algorithm delivers more messages than SPR algorithm.

4.4 Module Description

Server module

This module provides the part of the application that functions as a server by providing appropriate services to the clients. The server needs to identify the type of request made by the clients and will service the client's request. The server is in waiting state always to accept the requests. Files are transferred by the server to the client.

Client module

The client and server establish a communication path as defined in the application. The client will first decide the receiving path and then make a request to the server for the required file. The request is transferred and then the result of the request is obtained by the client.

Router module

The router module allows the application to set up a communication path between the client and server. The analysis of shortest path is done in this module with the help of the proposed algorithms. Based on our analytical results, we are able to show the precise relationship between

the number of nodes and the load at each node, and the geographical distribution of the relaying load over the network for different scenarios.

4.5 S/W & H/W Requirements

Software Requirements:

Operating System	:	Windows XP Professional
IDE	:	Visual Studio .Net 2008
Coding Language	:	C#.NET

Hardware Requirements:

Processor	:	Pentium IV 2.4 GHz
Hard Disk	:	160GB
Ram	:	1GB

5. SOFTWARE REQUIREMENT SPECIFICATION

5.1 Scope of the Project

The software, Site Explorer is designed for management of web sites from a remote location.

Purpose: The main purpose for preparing this document is to give a general insight into the analysis and requirements of the existing system or situation and for determining the operating characteristics of the system.

Scope: This Document plays a vital role in the development life cycle (SDLC) and it describes the complete requirement of the system. It is meant for use by the developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

Developers responsibilities overview:

The developer is responsible for:

- Developing the system, which meets the SRS and solving all the requirements of the system?
- Demonstrating the system and installing the system at client's location after the acceptance testing is successful.
- Submitting the required user manual describing the system interfaces to work on it and also the documents of the system.
- Conducting any user training that might be needed for using the system.
- Maintaining the system for a period of one year after installation.

5.2 Functional Requirements

Functional Requirements refer to very important system requirements in a software engineering process (or at micro level, a sub part of requirement engineering) such as technical specifications, system design parameters and guidelines, data manipulation, data processing and calculation modules etc.

Functional Requirements are in contrast to other software design requirements referred to as Non-Functional Requirements which are primarily based on parameters of system performance, software quality attributes, reliability and security, cost, constraints in design/implementation etc.

The key goal of determining "functional requirements" in a software product design and implementation is to capture the required behavior of a software system in terms of functionality and the technology implementation of the business processes.

The **Functional Requirement** document (also called Functional Specifications or Functional Requirement Specifications), defines the capabilities and functions that a System must be able to perform successfully.

Functional Requirements should include:

- Descriptions of data to be entered into the system
- Descriptions of operations performed by each screen
- Descriptions of work-flows performed by the system
- Descriptions of system reports or other outputs
- Who can enter the data into the system?
- How the system meets applicable regulatory requirements

The functional specification is designed to be read by a general audience. Readers should understand the system, but no particular technical knowledge should be required to understand the document.

Examples of Functional Requirements:

Functional requirements should include functions performed by specific screens, outlines of work-flows performed by the system and other business or compliance requirements the system must meet.

Interface requirements

- Field accepts numeric data entry
- Field only accepts dates before the current date

- Screen can print on-screen data to the printer

Business Requirements

- Data must be entered before a request can approved
- Clicking the Approve Button moves the request to the Approval Workflow
- All personnel using the system will be trained according to internal training strategies

Regulatory/Compliance Requirements

- The database will have a functional audit trail
- The system will limit access to authorized users
- The spreadsheet can secure data with electronic signatures

Security Requirements

- Member of the Data Entry group can enter requests but not approve or delete requests
- Members of the Managers group can enter or approve a request, but not delete requests
- Members of the Administrators group cannot enter or approve requests, but can delete requests

The functional specification describes what the system must do; how the system does it is described in the Design Specification.

If a User Requirement Specification was written, all requirements outlined in the user requirement specification should be addressed in the functional requirements.

5.3 Non Functional Requirements

All the other requirements which do not form a part of the above specification are categorized as Non-Functional Requirements.

A system may be required to present the user with a display of the number of records in a database. This is a functional requirement.

How up-to-date this number needs to be is a non-functional requirement. If the number needs to be updated in real time, the system architects must ensure that the system is capable of updating the displayed record count within an acceptably short interval of the number of records changing.

Sufficient network bandwidth may also be a non-functional requirement of a system.

Accessibility is a general term used to describe the degree to which a product, device, service, or environment is accessible by as many people as possible. Accessibility can be viewed as the "ability to access" and possible benefit of some system or entity. Accessibility is often used to focus on people with disabilities and their right of access to the system.

Availability is the degree to which a system, subsystem, or equipment is operable and in a committable state at the start of a mission, when the mission is called for at an unknown, *i.e.,* a random, time. Simply put, availability is the proportion of time a system is in a functioning condition. Expressed mathematically, **availability** is 1 minus the unavailability.

A **backup** or the process of **backing up** refers to making copies of data so that these additional copies may be used to *restore* the original after a data loss event. These additional copies are typically called "backups."

Certification refers to the confirmation of certain characteristics of an object, system, or organization. This confirmation is often, but not always, provided by some form of external review, education, or assessment

6. SYSTEM DESIGN

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement have been specified and analyzed, system design is the first of the three technical activities -design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system – one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities – architectural design, data structure design, interface design and procedural design.

6.1 Normalization

It is a process of converting a relation to a standard form. The process is used to handle the problems that can arise due to data redundancy i.e. repetition of data in the database, maintain data integrity as well as handling problems that can arise due to insertion, updating, deletion anomalies.

Decomposing is the process of splitting relations into multiple relations to eliminate anomalies and maintain anomalies and maintain data integrity. To do this we use normal forms or rules for structuring relation.

Insertion anomaly: Inability to add data to the database due to absence of other data.

Deletion anomaly: Unintended loss of data due to deletion of other data.

Update anomaly: Data inconsistency resulting from data redundancy and partial update

Normal Forms: These are the rules for structuring relations that eliminate anomalies.

First normal form:

A relation is said to be in first normal form if the values in the relation are atomic for every attribute in the relation. By this we mean simply that no attribute value can be a set of values or, as it is sometimes expressed, a repeating group.

Second normal form:

A relation is said to be in second Normal form is it is in first normal form and it should satisfy any one of the following rules.

1) Primary key is a not a composite primary key
2) No non key attributes are present
3) Every non key attribute is fully functionally dependent on full set of primary key.

Third normal form:

A relation is said to be in third normal form if their exits no transitive dependencies.

Transitive Dependency: If two non key attributes depend on each other as well as on the primary key then they are said to be transitively dependent.

The above normalization principles were applied to decompose the data in multiple tables thereby making the data to be maintained in a consistent state.

6.2 Data Flow Diagrams

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams. The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose. The development of DFD'S is done in several levels. Each process in

lower level diagrams can be broken down into a more detailed DFD in the next level. The lop-level diagram is often called context diagram. It consists a single process bit, which plays vital role in studying the current system. The process in the context level diagram is exploded into other process at the first level DFD.

The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

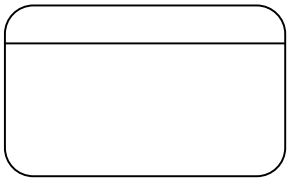
Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical from, this lead to the modular design.

A DFD is also known as a "bubble Chart" has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

6.2.1 DFD Symbols:

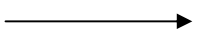
In the DFD, there are four symbols

1. A square defines a source(originator) or destination of system data
2. An arrow identifies data flow. It is the pipeline through which the information flows
3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
4. An open rectangle is a data store, data at rest or a temporary repository of data

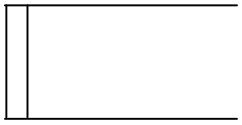


Process that transforms data flow.

Source or Destination of data



Data flow



Data Store

6.2.2 Constructing a DFD:

Several rules of thumb are used in drawing DFD'S:

1. Process should be named and numbered for an easy reference. Each name should be representative of the process.
2. The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.
3. When a process is exploded into lower level details, they are numbered.
4. The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each work capitalized.

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out. Missing interfaces redundancies and like is then accounted for often through interviews.

Salient Features Of DFD's

1. The DFD shows flow of data, not of control loops and decision are controlled considerations do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the dataflow take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

Types of Data Flow Diagrams

1. Current Physical

2. Current Logical
3. New Logical
4. New Physical

Current physical

In Current Physical DFD process label include the name of people or their positions or the names of computer systems that might provide some of the overall system-processing label includes an identification of the technology used to process the data. Similarly data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business forms or computer tapes.

Current logical

The physical aspects at the system are removed as much as possible so that the current system is reduced to its essence to the data and the processors that transforms them regardless of actual physical form.

New logical:

This is exactly like a current logical model if the user were completely happy with the user were completely happy with the functionality of the current system but had problems with how it was implemented typically through the new logical model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

New physical:

The new physical represents only the physical implementation of the new system.

Rules Governing the DFD's

Process

1) No process can have only outputs.
2) No process can have only inputs. If an object has only inputs than it must be a sink.
3) A process has a verb phrase label.

Data Store

1) Data cannot move directly from one data store to another data store, a process must move data.
2) Data cannot move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store
3) A data store has a noun phrase label.

Source or Sink

The origin and /or destination of data.

1) Data cannot move direly from a source to sink it must be moved by a process
2) A source and /or sink has a noun phrase land

Data Flow

1) A Data Flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The later is usually indicated however by two separate arrows since these happen at different type.
2) A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.
3) A data flow cannot go directly back to the same process it leads. There must be at least one other process that handles the data flow produce some other data flow returns the original data into the beginning process.
4) A Data flow to a data store means update (delete or change).
5) A data Flow from a data store means retrieve or use.

A data flow has a noun phrase label more than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

Data Flow Diagram

DFD:

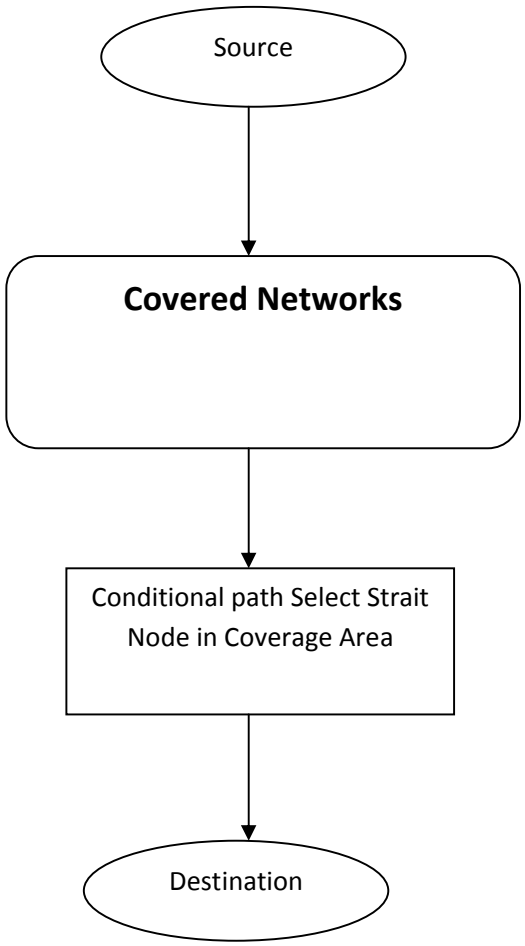
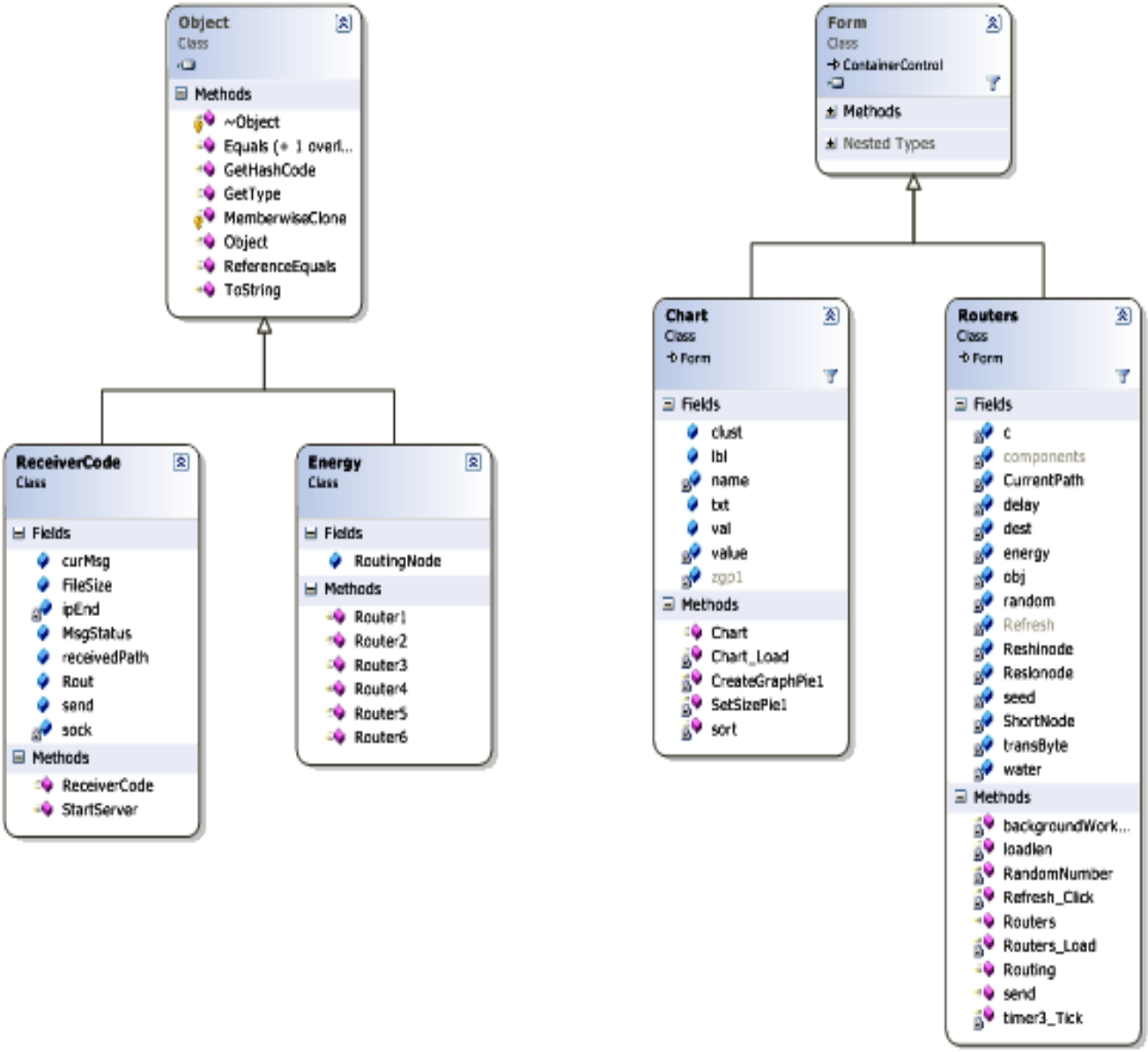


Fig-7 Data Flow Diagram

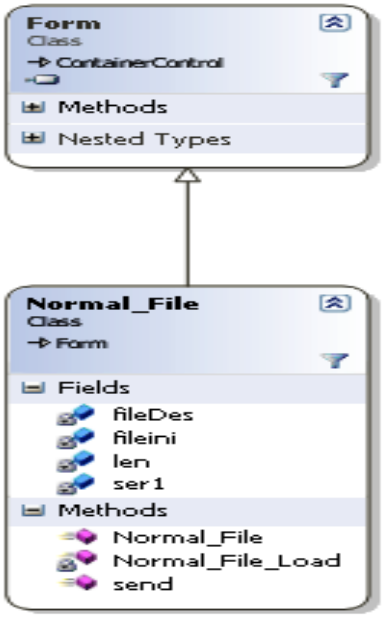
6.3 UML Diagrams

6.3.1 Class Diagram:

Router:



Server:



Client:

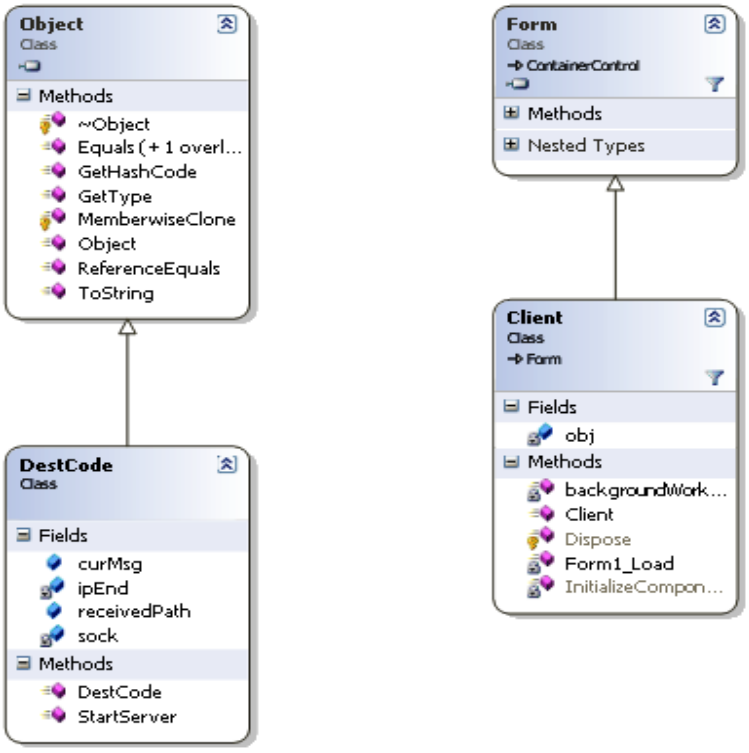


Fig-8. Class Diagram

6.3.2 Use case Diagram:

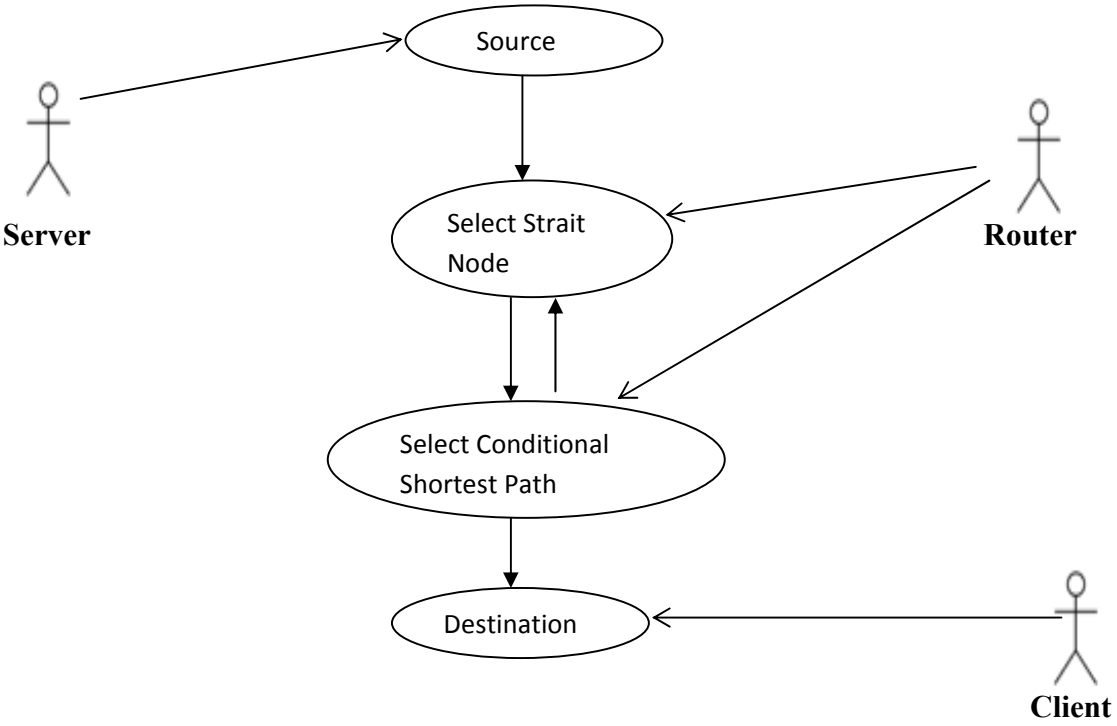


Fig-9. Use Case Diagram

6.3.3 Sequence Diagram:

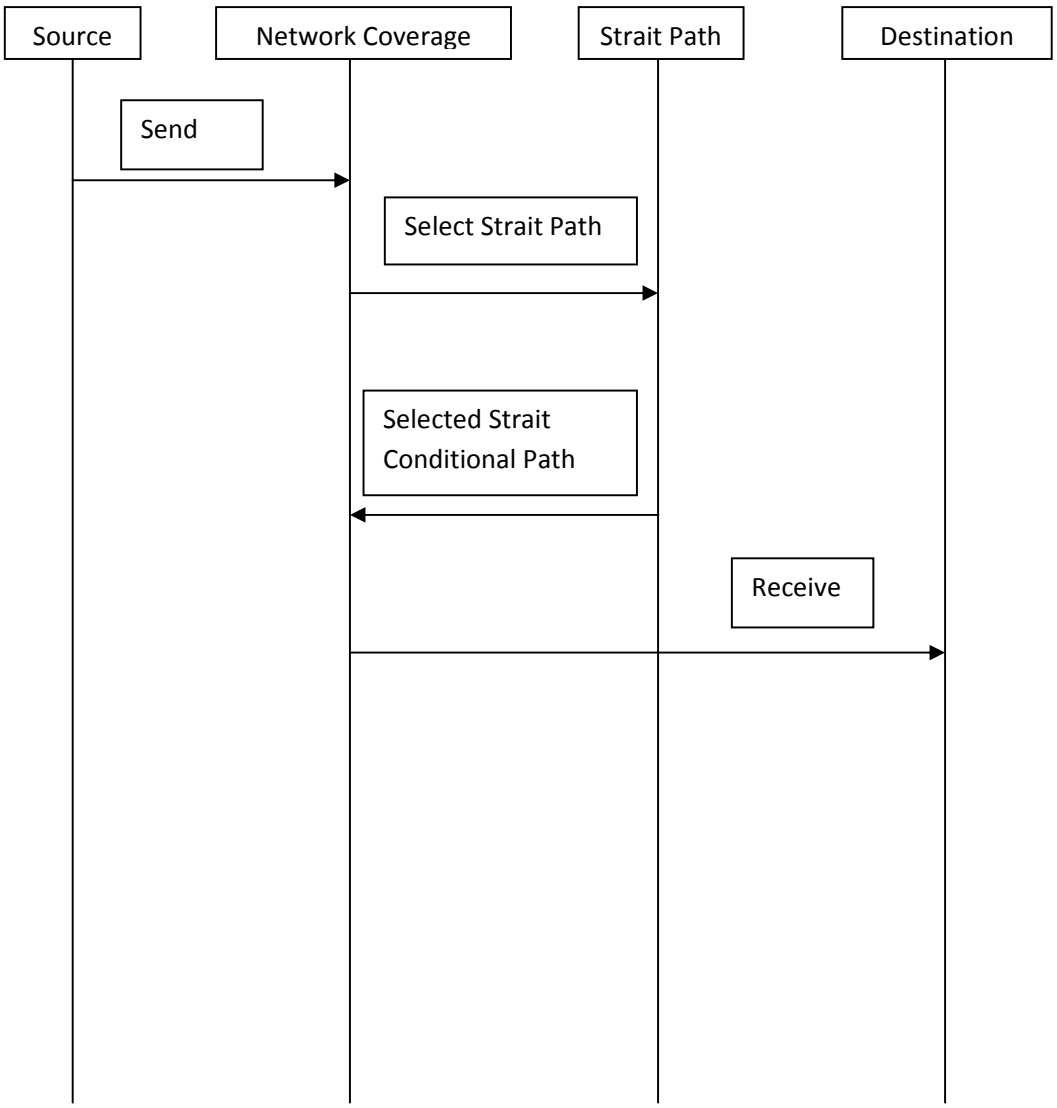


Fig-10. Sequence Diagram

6.3.4 Activity:

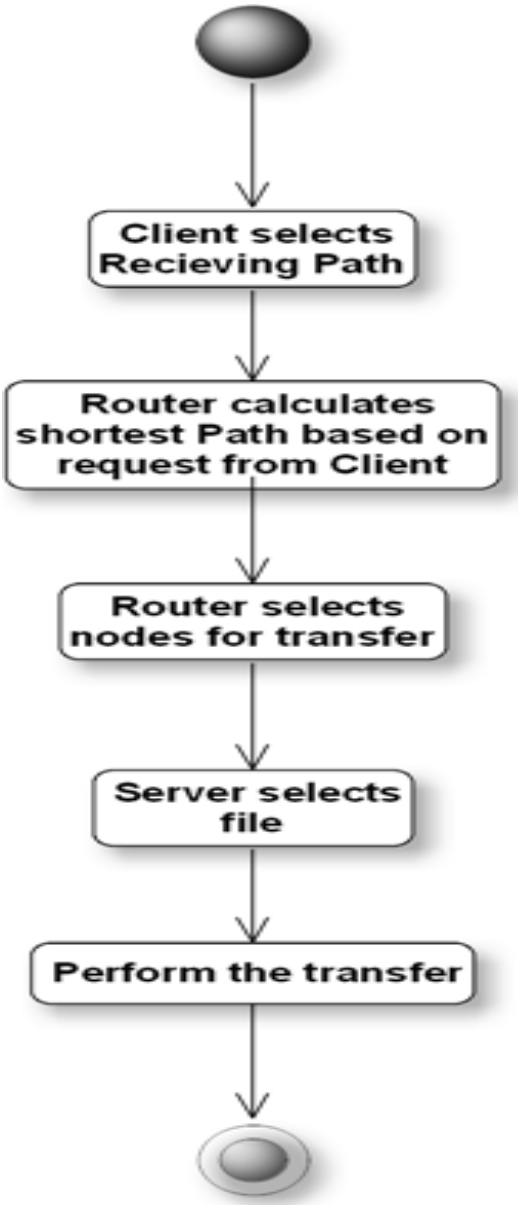


Fig-11. Activity Diagram

6.3.5 Collaboration Diagram

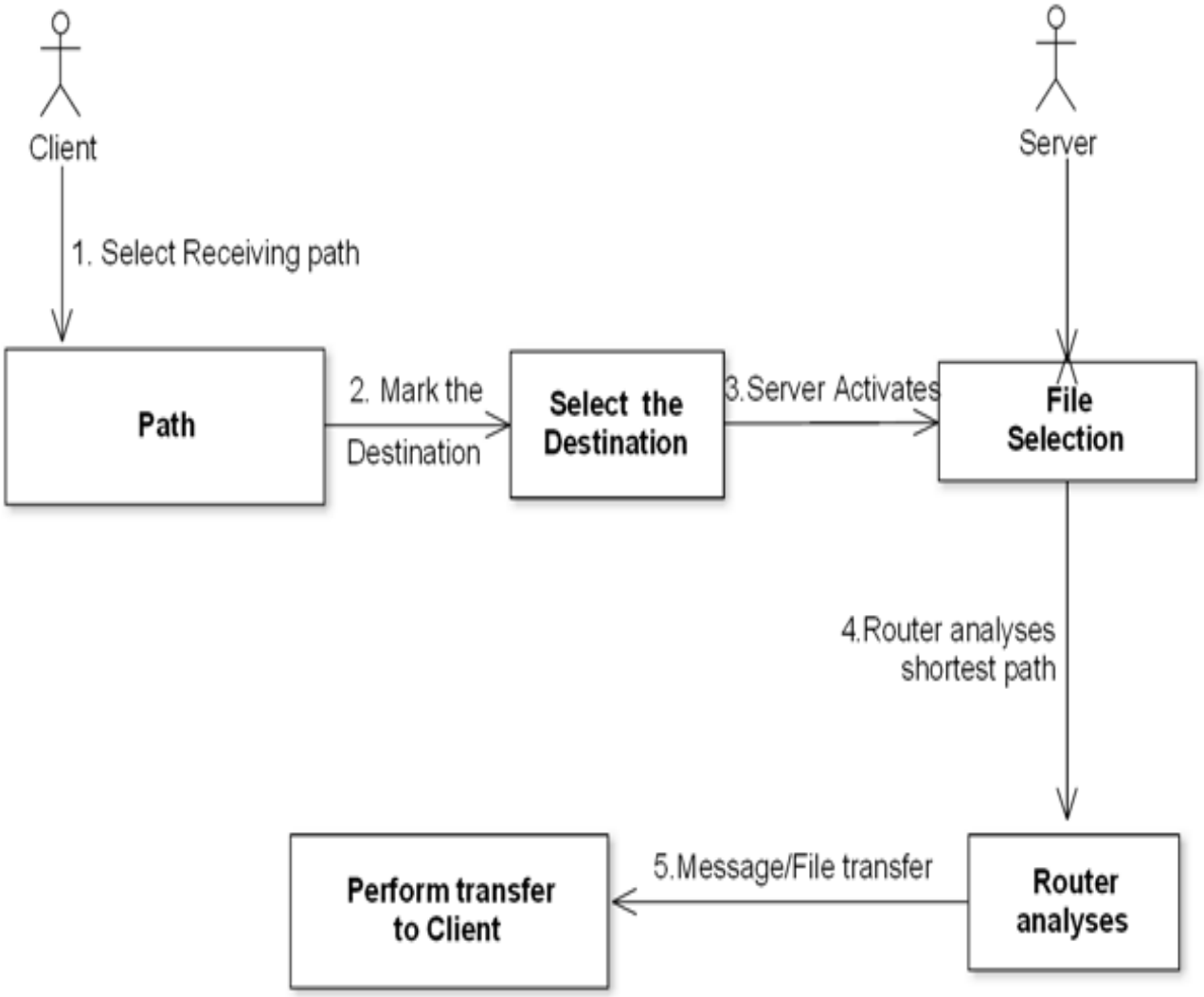


Fig-12. Collaboration Diagram

7. TECHNOLOGY DESCRIPTION

7.1. Net Framework

7.1.1 Introduction

The **Microsoft .NET Framework** is a software technology that is available with several Microsoft Windows operating systems. It includes a large library of pre-coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering and is intended to be used by most new applications created for the Windows platform.

The pre-coded solutions that form the framework's Base Class Library cover a large range of programming needs in a number of areas, including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by programmers, who combine it with their own code to produce applications.

Programs written for the .NET Framework execute in a software environment that manages the program's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and exception handling. The class library and the CLR together compose the .NET Framework.

7.1.2 Principal design features

Interoperability

Because interaction between new and older applications is commonly required, the .NET Framework provides means to access functionality that is implemented in programs that execute outside the .NET environment. Access to COM components is provided in the System Runtime, InteropServices and System Enterprise Services namespaces of the framework; access to other functionality is provided using the P/Invoke feature.

Common Runtime Engine

The Common Language Runtime (CLR) is the virtual machine component of the .NET framework. All .NET programs execute under the supervision of the CLR, guaranteeing certain properties and behaviors in the areas of memory management, security, and exception handling.

Base Class Library

The Base Class Library (BCL), part of the Framework Class Library (FCL), is a library of functionality available to all languages using the .NET Framework. The BCL provides classes which encapsulate a number of common functions, including file reading and writing, graphic rendering, database interaction and XML document manipulation.

Simplified Deployment

Installation of computer software must be carefully managed to ensure that it does not interfere with previously installed software, and that it conforms to security requirements. The .NET framework includes design features and tools that help address these requirements.

Security

The design is meant to address some of the vulnerabilities, such as buffer overflows, that have been exploited by malicious software. Additionally, .NET provides a common security model for all applications.

Portability

The design of the .NET Framework allows it to theoretically be platform agnostic, and thus cross-platform compatible. That is, a program written to use the framework should run without change on any type of system for which the framework is implemented. Microsoft's commercial implementations of the framework cover Windows, Windows CE, and the Xbox 360. In addition, Microsoft submits the specifications for the Common Language Infrastructure (which includes the core class libraries, Common Type System, and the Common Intermediate Language), the C# language, and the C++/CLI language to both ECMA and the ISO, making them available as open standards. This makes it possible for third parties to create compatible implementations of the framework and its languages on other platforms.

7.1.3 Architecture

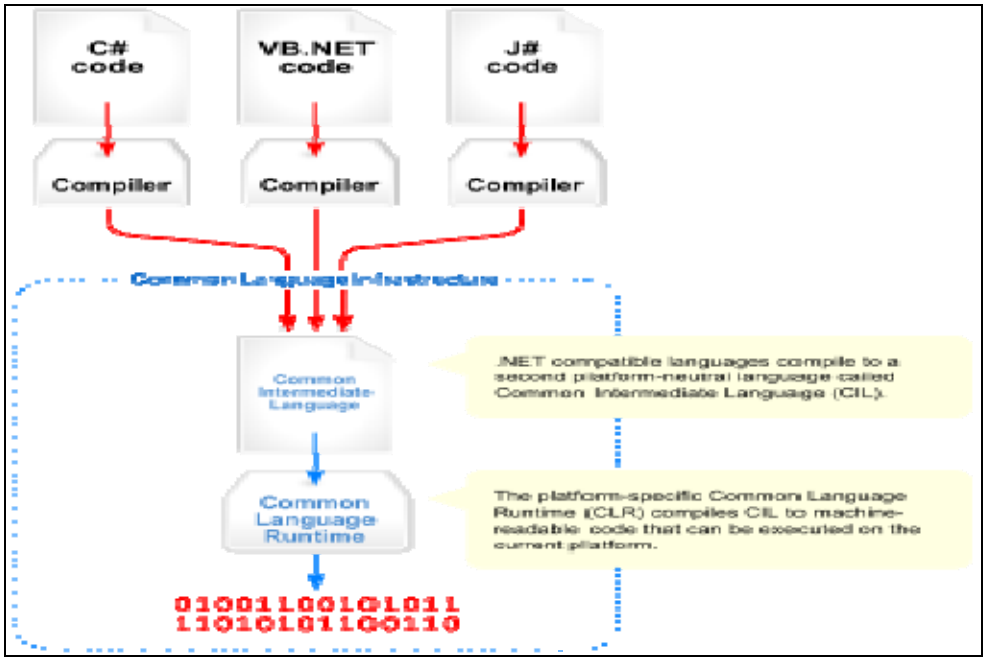


Fig-13. Visual overview of the Common Language Infrastructure (CLI)

Common Language Infrastructure

The core aspects of the **.NET framework** lie within the Common Language Infrastructure, or **CLI**. The purpose of the CLI is to provide a language-neutral platform for application development and execution, including functions for exception handling, garbage collection, security, and interoperability. Microsoft's implementation of the CLI is called the **Common Language Runtime** or **CLR**.

Assemblies

The intermediate CIL code is housed in .NET assemblies. As mandated by specification, assemblies are stored in the Portable Executable (PE) format, common on the Windows platform for all DLL and EXE files. The assembly consists of one or more files, one of which must contain the manifest, which has the metadata for the assembly. The complete name of an assembly (not to be confused with the filename on disk) contains its simple text name, version number, culture, and public key token. The public key token is a unique hash generated when the assembly is compiled, thus two assemblies with the same public key token are guaranteed to be identical from the point of view of the framework. A private key can also be specified known only to the creator of the

assembly and can be used for strong naming and to guarantee that the assembly is from the same author when a new version of the assembly is compiled (required to add an assembly to the Global Assembly Cache).

Metadata

All CLI is self-describing through .NET metadata. The CLR checks the metadata to ensure that the correct method is called. Metadata is usually generated by language compilers but developers can create their own metadata through custom attributes. Metadata contains information about the assembly, and is also used to implement the reflective programming capabilities of .NET Framework.

Security

.NET has its own security mechanism with two general features: Code Access Security (CAS), and validation and verification. Code Access Security is based on evidence that is associated with a specific assembly. Typically the evidence is the source of the assembly (whether it is installed on the local machine or has been downloaded from the intranet or Internet). Code Access Security uses evidence to determine the permissions granted to the code. Other code can demand that calling code is granted a specified permission. The demand causes the CLR to perform a call stack walk: every assembly of each method in the call stack is checked for the required permission; if any assembly is not granted the permission a security exception is thrown.

When an assembly is loaded the CLR performs various tests. Two such tests are validation and verification. During validation the CLR checks that the assembly contains valid metadata and CIL, and whether the internal tables are correct. Verification is not so exact. The verification mechanism checks to see if the code does anything that is 'unsafe'. The algorithm used is quite conservative; hence occasionally code that is 'safe' does not pass. Unsafe code will only be executed if the assembly has the 'skip verification' permission, which generally means code that is installed on the local machine.

.NET Framework uses appdomains as a mechanism for isolating code running in a process. Appdomains can be created and code loaded into or unloaded from them independent of other appdomains. This helps increase the fault tolerance of the application, as faults or crashes in one appdomain do not affect rest of the application. Appdomains can also be configured independently

with different security privileges. This can help increase the security of the application by isolating potentially unsafe code. The developer, however, has to split the application into sub domains; it is not done by the CLR.

Class library

Namespaces in the BCL
System
System. CodeDom
System. Collections
System. Diagnostics
System. Globalization
System. IO
System. Resources
System. Text
System.Text.RegularExpressions

Microsoft **.NET Framework** includes a set of standard **class libraries**. The class library is organized in a hierarchy of namespaces. Most of the built in APIs are part of either System.* or Microsoft.* namespaces. It encapsulates a large number of common functions, such as file reading and writing, graphic rendering, database interaction, and XML document manipulation, among others. The .NET class libraries are available to all .NET languages. The .NET Framework class library is divided into two parts: the **Base Class Library** and the **Framework Class Library**.

The **Base Class Library** (BCL) includes a small subset of the entire class library and is the core set of classes that serve as the basic API of the Common Language Runtime. The classes in mscorlib.dll and some of the classes in System.dll and System.core.dll are considered to be a part of the BCL. The BCL classes are available in both .NET Framework as well as its alternative implementations including .NET Compact Framework, Microsoft Silver light and Mono.

The **Framework Class Library** (FCL) is a superset of the BCL classes and refers to the entire class library that ships with .NET Framework. It includes an expanded set of libraries,

including Win Forms, ADO.NET, ASP.NET, Language Integrated Query, Windows Presentation Foundation, Windows Communication Foundation among others. The FCL is much larger in scope than standard libraries for languages like C++, and comparable in scope to the standard libraries of Java.

Memory management

The .NET Framework CLR frees the developer from the burden of managing memory (allocating and freeing up when done); instead it does the memory management itself. To this end, the memory allocated to instantiations of .NET types (objects) is done contiguously from the managed heap, a pool of memory managed by the CLR. As long as there exists a reference to an object, which might be either a direct reference to an object or via a graph of objects, the object is considered to be in use by the CLR. When there is no reference to an object, and it cannot be reached or used, it becomes garbage. However, it still holds on to the memory allocated to it. .NET Framework includes a garbage collector which runs periodically, on a separate thread from the application's thread, that enumerates all the unusable objects and reclaims the memory allocated to them.

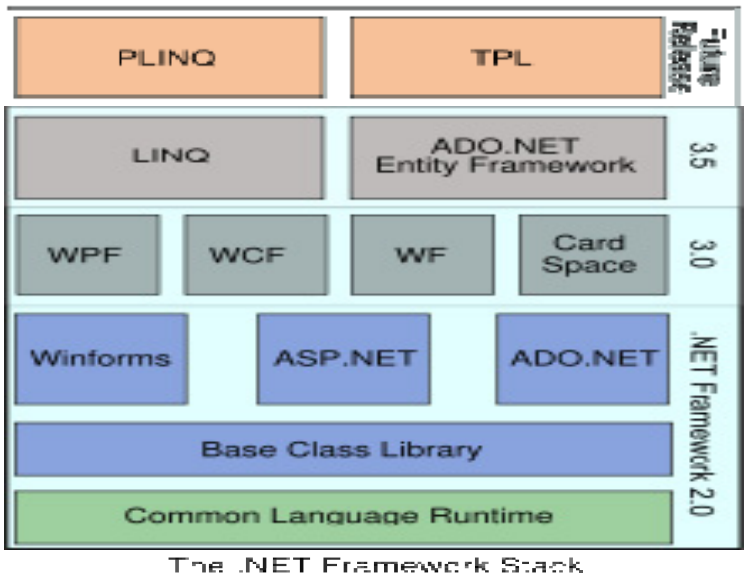
The .NET Garbage Collector (GC) is a non-deterministic, compacting, mark-and-sweep garbage collector. The GC runs only when a certain amount of memory has been used or there is enough pressure for memory on the system. Since it is not guaranteed when the conditions to reclaim memory are reached, the GC runs are non-deterministic. Each .NET application has a set of roots, which are pointers to objects on the managed heap (*managed objects*). These include references to static objects and objects defined as local variables or method parameters currently in scope, as well as objects referred to by CPU registers. When the GC runs, it pauses the application, and for each object referred to in the root, it recursively enumerates all the objects reachable from the root objects and marks them as reachable. It uses .NET metadata and reflection to discover the objects encapsulated by an object, and then recursively walk them. It then enumerates all the objects on the heap (which were initially allocated contiguously) using reflection. All objects not marked as reachable are garbage. This is the *mark* phase. Since the memory held by garbage is not of any consequence, it is considered free space. However, this leaves chunks of free space between objects which were initially contiguous. The objects are then *compacted* together, by using memory to copy them over to the free space to make them

contiguous again. Any reference to an object invalidated by moving the object is updated to reflect the new location by the GC. The application is resumed after the garbage collection is over.

The GC used by .NET Framework is actually *generational*. Objects are assigned a *generation*; newly created objects belong to *Generation 0*. The objects that survive a garbage collection are tagged as *Generation 1*, and the Generation 1 objects that survive another collection are *Generation 2* objects. The .NET Framework uses up to Generation 2 objects. Higher generation objects are garbage collected less frequently than lower generation objects. This helps increase the efficiency of garbage collection, as older objects tend to have a larger lifetime than newer objects. Thus, by removing older (and thus more likely to survive a collection) objects from the scope of a collection run, fewer objects need to be checked and compacted.

Versions

Microsoft started development on the .NET Framework in the late 1990s originally under the name of Next Generation Windows Services (NGWS). By late 2000 the first beta versions of .NET 1.0 were released.



The .NET Framework Stack

Fig-14. **.NET Framework** Architecture

Version	Version Number	Release Date
1.0	1.0.3705.0	2002-01-05
1.1	1.1.4322.573	2003-04-01
2.0	2.0.50727.42	2005-11-07
3.0	3.0.4506.30	2006-11-06
3.5	3.5.21022.8	2007-11-09

7.2 C#.NET

Ado.Net Overview

ADO.NET is an evolution of the ADO data access model that directly addresses user requirements for developing scalable applications. It was designed specifically for the web with scalability, statelessness, and XML in mind.

ADO.NET uses some ADO objects, such as the **Connection** and **Command** objects, and also introduces new objects. Key new ADO.NET objects include the **Dataset**, **Data Reader**, and **Data Adapter**.

The important distinction between this evolved stage of ADO.NET and previous data architectures is that there exists an object -- the **DataSet** -- that is separate and distinct from any data stores. Because of that, the **DataSet** functions as a standalone entity. You can think of the DataSet as an always disconnected recordset that knows nothing about the source or destination of the data it contains. Inside a **DataSet**, much like in a database, there are tables, columns, relationships, constraints, views, and so forth.

A **DataAdapter** is the object that connects to the database to fill the **DataSet**. Then, it connects back to the database to update the data there, based on operations performed while the **DataSet** held the data. In the past, data processing has been primarily connection-based. Now, in an effort to make multi-tiered apps more efficient, data processing is turning to a message-based approach that revolves around chunks of information. At the center of this approach is the **DataAdapter**, which provides a bridge to retrieve and save data between a **DataSet** and its source data store. It accomplishes this by means of requests to the appropriate SQL commands made against the data store.

The XML-based **DataSet** object provides a consistent programming model that works with all models of data storage: flat, relational, and hierarchical. It does this by having no 'knowledge' of the source of its data, and by representing the data that it holds as collections and data types. No matter what the source of the data within the **DataSet** is, it is manipulated through the same set of standard APIs exposed through the **DataSet** and its subordinate objects.

While the **DataSet** has no knowledge of the source of its data, the managed provider has detailed and specific information. The role of the managed provider is to connect, fill, and persist the **DataSet** to and from data stores. The OLE DB and SQL Server .NET Data Providers (System.Data.OleDb and System.Data.SqlClient) that are part of the .Net Framework provide four basic objects: the **Command**, **Connection**, **DataReader** and **DataAdapter**. In the remaining sections of this document, we'll walk through each part of the **DataSet** and the OLE DB/SQL Server .NET Data Providers explaining what they are, and how to program against them.
The following sections will introduce you to some objects that have evolved, and some that are new. These objects are:

- **Connections**. For connection to and managing transactions against a database.
- **Commands**. For issuing SQL commands against a database.
- **DataReaders**. For reading a forward-only stream of data records from a SQL Server data source.
- **DataSet**. For storing, Remoting and programming against flat data, XML data and relational data.
- **DataAdapters**. For pushing data into a **DataSet**, and reconciling data against a database.

When dealing with connections to a database, there are two different options: SQL Server .NET Data Provider (System.Data.SqlClient) and OLE DB .NET Data Provider (System.Data.OleDb). In these samples we will use the SQL Server .NET Data Provider. These are written to talk directly to Microsoft SQL Server. The OLE DB .NET Data Provider is used to talk to any OLE DB provider (as it uses OLE DB underneath).

Connections:

Connections are used to 'talk to' databases, and are represented by provider-specific classes such as **SqlConnection**. Commands travel over connections and resultsets are returned in the form of streams which can be read by a **DataReader** object, or pushed into a **DataSet** object.

Commands:

Commands contain the information that is submitted to a database, and are represented by provider-specific classes such as **SqlCommand**. A command can be a stored procedure call, an UPDATE statement, or a statement that returns results. You can also use input and output parameters, and return values as part of your command syntax. The example below shows how to issue an INSERT statement against the **Northwind** database.

DataReaders:

The **Data Reader** object is somewhat synonymous with a read-only/forward-only cursor over data. The **DataReader** API supports flat as well as hierarchical data. A **DataReader** object is returned after executing a command against a database. The format of the returned **DataReader** object is different from a recordset. For example, you might use the **DataReader** to show the results of a search list in a web page.

Datasets and dataadapters:

DataSets

The **Dataset** object is similar to the ADO **Recordset** object, but more powerful, and with one other important distinction: the **DataSet** is always disconnected. The **DataSet** object represents a cache of data, with database-like structures such as tables, columns, relationships, and constraints. However, though a **DataSet** can and does behave much like a database, it is important to remember that **DataSet** objects do not interact directly with databases, or other source data. This allows the developer to work with a programming model that is always consistent, regardless of where the source data resides. Data coming from a database, an XML file, from code, or user input can all be placed into **DataSet** objects. Then, as changes are made to the **DataSet** they can be tracked and verified before updating the source data. The **GetChanges** method of the **DataSet** object actually creates a second **DatSet** that contains only the changes to the data. This **DataSet** is then used by a **DataAdapter** (or other objects) to update the original data source.

The **DataSet** has many XML characteristics, including the ability to produce and consume XML data and XML schemas. XML schemas can be used to describe schemas interchanged via WebServices. In fact, a **DataSet** with a schema can actually be compiled for type safety and statement completion.

Dataadapters (Oledb/Sql)

The **DataAdapter** object works as a bridge between the **DataSet** and the source data. Using the provider-specific **SqlDataAdapter** (along with its associated **SqlCommand** and **SqlConnection**) can increase overall performance when working with a Microsoft SQL Server databases. For other OLE DB-supported databases, you would use the **OleDbDataAdapter** object and its associated **OleDbCommand** and **OleDbConnection** objects.

The **DataAdapter** object uses commands to update the data source after changes have been made to the **DataSet**. Using the **Fill** method of the **DataAdapter** calls the SELECT command; using the **Update** method calls the INSERT, UPDATE or DELETE command for each changed row. You can explicitly set these commands in order to control the statements used at runtime to resolve changes, including the use of stored procedures. For ad-hoc scenarios, a **CommandBuilder** object can generate these at run-time based upon a select statement. However, this run-time generation requires an extra round-trip to the server in order to gather required metadata, so explicitly providing the INSERT, UPDATE, and DELETE commands at design time will result in better run-time performance.

1. ADO.NET is the next evolution of ADO for the .Net Framework.
2. ADO.NET was created with n-Tier, statelessness and XML in the forefront. Two new objects, the **DataSet** and **DataAdapter**, are provided for these scenarios.
3. ADO.NET can be used to get data from a stream, or to store data in a cache for updates.
4. There is a lot more information about ADO.NET in the documentation.
5. Remember, you can execute a command directly against the database in order to do inserts, updates, and deletes. You don't need to first put data into a **DataSet** in order to insert, update, or delete it.

Also, you can use a **DataSet** to bind to the data, move through the data, and navigate data relationships

8. IMPLEMENTATION

Client Module Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
namespace DestCode
{ public partial class Client : Form
    { public Client()
        {   InitializeComponent();
            DestCode.receivedPath = "";     }
        private void timer1_Tick(object sender, EventArgs e)
        {   label5.Text = DestCode.receivedPath;
            lblres.Text = DestCode.curMsg;    }
        DestCode obj = new DestCode();
       private void backgroundWorker1_DoWork(object sender,DoWorkEventArgs e)
        {      obj.StartServer();  }
        private void Form1_Load(object sender, EventArgs e)
        {  backgroundWorker1.RunWorkerAsync();  }
        private void button2_Click(object sender, EventArgs e)
        {  FolderBrowserDialog f = new FolderBrowserDialog();
            f.ShowDialog();
            if (f.SelectedPath != "")      {
                DestCode.receivedPath = f.SelectedPath;
```

```
            label5.Text = f.SelectedPath;       }
        else
              { MessageBox.Show("Please Select a File Receiving Path.\r\n
              Else You Can not Receive the File");
     } }
}//FILE TRANSFER USING C#.NET SOCKET - SERVER
class DestCode
{  IPEndPoint ipEnd;
    Socket sock;
    public DestCode()
    {  IPHostEntry ipEntry = Dns.GetHostEntry(Environment.MachineName);
        IPAddress IpAddr = ipEntry.AddressList[0];
        ipEnd = new IPEndPoint(IpAddr, 5656);
        sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
        ProtocolType.IP);
        sock.Bind(ipEnd);      }
    public static string receivedPath;
    public static string curMsg = "Stopped";
    public  void StartServer()
    {  try
        {   sock.Listen(100);
            Socket clientSock = sock.Accept();
            byte[] clientData = new byte[1024 * 5000];
            int receivedBytesLen = clientSock.Receive(clientData);
            curMsg = "Receiving data...";
            int fileNameLen = BitConverter.ToInt32(clientData, 0);
            string  fileName  =  Encoding.ASCII.GetString(clientData,  4,
            fileNameLen);
            BinaryWriter bWrite = new BinaryWriter(File.Open(receivedPath
            +"/"+ fileName, FileMode.Append)); ;
            bWrite.Write(clientData,4 + fileNameLen, receivedBytesLen - 4 -
            fileNameLen);
            if (receivedPath == "")
        { MessageBox.Show("No Path was selected to Save the File");   }
```

```
                curMsg = "Saving file...";
                bWrite.Close();
                clientSock.Close();
                curMsg = "File Received ...";
                StartServer();       }
            catch(Exception ex) {
        curMsg = "File Receving error.";     }    }
    }
}
```

Server Module Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Net.Sockets;
using System.Net;
using System.IO;
using System.Globalization;
namespace Source
{    public partial class Normal_File : Form
    {   string ser1;
        string fileDes, fileini;
        int len;
        public Normal_File()    {
         InitializeComponent(); }
        private void Normal_File_Load(object sender, EventArgs e)
        {  btnSend.Enabled = false;
```

```
    this.openFileDialog1.Multiselect = true;  }
 private void btnOpen_Click(object sender, EventArgs e)
 {   txtFilePath.Text = "";
     openFileDialog1.ShowDialog();
     txtFilePath.Text = openFileDialog1.FileName;
     fileDes = openFileDialog1.FileName;
     if (fileDes == "openFileDialog1")   {
         lblError.Text = "";
         lblError.Text = "Select a File first";
         txtFilePath.Text = "";
         btnSend.Enabled = false;   }
     else
     {  len = fileDes.Length;
        fileini = fileDes.Substring(fileDes.IndexOf("\\") + 1);
        btnSend.Enabled = true;    }
 }
private void btnSend_Click(object sender, EventArgs e)
 {   pic1.Visible = true;
     Application.DoEvents();
     send();    }
 public void send()
 {   try
     {   IPAddress[] ipAddress = Dns.GetHostAddresses(txtIp.Text);
         IPEndPoint ipEnd = new IPEndPoint(ipAddress[0], 5655);
           Socket clientSock = new Socket(AddressFamily.InterNetwork,
           SocketType.Stream, ProtocolType.IP);
         string filePath = "";
         int count = 0;
         fileDes = fileDes.Replace("\\", "/");
         while (fileDes.IndexOf("/") > -1)
         { filePath += fileDes.Substring(0, fileDes.IndexOf("/") + 1);
           fileDes = fileDes.Substring(fileDes.IndexOf("/") + 1);
```

```
                count++; }

            byte[] fileNameByte = Encoding.ASCII.GetBytes(fileDes);

            lblError.Text = "";

            lblError.Text = "Buffering ...";

            byte[] fileData = File.ReadAllBytes(filePath + fileDes);

               byte[] clientData = new byte[4 + fileNameByte.Length +
              fileData.Length];

        byte[] fileNameLen = BitConverter.GetBytes(fileNameByte.Length);


            fileNameLen.CopyTo(clientData, 0);

            fileNameByte.CopyTo(clientData, 4);

            fileData.CopyTo(clientData, 4 + fileNameByte.Length);

            lblError.Text = "";

            lblError.Text = "Connection to server ...";

            clientSock.Connect(ipEnd);

            lblError.Text = "";

            lblError.Text = "File sending...";

            System.Threading.Thread.Sleep(1000);

            clientSock.Send(clientData);

            lblError.Text = "";

            lblError.Text = "Disconnecting...";

            clientSock.Close();

            lblError.Text = "";

            lblError.Text = "File transferred.";      }
 catch (Exception ex)

       {   if (ex.Message == "A connection attempt failed because the connected
      party did not properly respond after a period of time, or established
      connection failed because connected host has failed to respond")

         { lblError.Text = "";

            lblError.Text = "No Such System Available Try other IP";     }

      else {

if (ex.Message == "No connection could be made because the target machine
actively refused it")
```

```
{  lblError.Text = "";
   lblError.Text = "File Sending fail. Because server not running.";
}
   else
     {  lblError.Text = "";
        lblError.Text = "File Sending fail." + ex.Message;    }
     }  }  }
private void button1_Click(object sender, EventArgs e)
   { this.openFileDialog1.Multiselect = true;
     DialogResult dr = this.openFileDialog1.ShowDialog();
     if (dr == System.Windows.Forms.DialogResult.OK)  {
  foreach (string file in openFileDialog1.FileNames)   {
            txtFilePath.Text += file;
            fileDes += file;
           if (fileDes == "openFileDialog1")    {
                     lblError.Text = "";
                     lblError.Text = "Select a File first";
                     txtFilePath.Text = "";
                     btnSend.Enabled = false;    }
                else
                { len = fileDes.Length;
                  fileini = fileDes.Substring(fileDes.IndexOf("\\") + 1);
                    btnSend.Enabled = true;  }    }
        }}}}
```

Router Module Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
```

```
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Net.Sockets;
using System.Net;
using System.IO;
namespace MC902
{  public partial class Routers : Form
    {   Energy energy = new Energy();
        Chart c = new Chart();
        public Routers()    {
        InitializeComponent();  }
        int delay = 0; int i = 0; byte[] dest; bool seed, water;
        String[]  ShortNode; String[] transByte;
        Random random = new Random();
        int[] Reshinode = new int[5]; int[] Reslonode = new int[5];
        string rn1 = "Router 1";
        string rn2 = "Router 2";
        string rn3 = "Router 3";
        string rn4 = "Router 4";
        string rn5 = "Router 5";
        string rn6 = "Router 6";
        string rn7 = "Router 7";
        string rn8 = "Router 8";
        string rn9 = "Router 9";
        string rn10 = "Router 10";
        string rn11 = "Router 11";
        string rn12 = "Router 12";
        string rn13 = "Router 13";
        string rn14 = "Router 14";
        string rn15 = "Router 15";
        string CurrentPath;
```

```
  private int RandomNumber()  {
      return random.Next(100, 200);   }
  ReceiverCode obj = new ReceiverCode();
  private void Routers_Load(object sender, EventArgs e)
  {   backgroundWorker1.RunWorkerAsync();
      seed = true;
      loadlen();   }
  private void loadlen()
  {  while (false)  {    }
  }
  public void Routing()
  { if (N1.Checked == true)   {
          p1.Image = MC902.Properties.Resources.Green11;
          Application.DoEvents();
          ShortNode[i] = rn1 ;
transByte[i] = Convert.ToString(ReceiverCode.send.Length ) + "(Bits)";
          i++;
      rn1 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
      "(Bits) Transmited";
          System.Threading.Thread.Sleep(1000);
          p6.Image = MC902.Properties.Resources.Green11;
          Application.DoEvents();
          ShortNode[i] = rn6;
 ansByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
          i++;
      rn6 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
      "(Bits) Transmited";
          System.Threading.Thread.Sleep(1000);
          for (int j = 0; j < ShortNode.Length; j++)
          {   CurrentPath += ShortNode[j].ToString();
              CurrentPath += "==#>";     }
          lblpath.Text = CurrentPath;
```

```
            send();       }
        else if (N2.Checked == true)
        {   p1.Image = MC902.Properties.Resources.Green11;
            Application.DoEvents();
            ShortNode[i] = rn1;
            transByte[i] = ReceiverCode.send.ToString() + "(Bits)";
            i++;
        rn1 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";
            System.Threading.Thread.Sleep(1000);

            p5.Image = MC902.Properties.Resources.Green11;
            Application.DoEvents();
            ShortNode[i] = rn5;
    transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
            i++;
         rn5 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";
            System.Threading.Thread.Sleep(1000);

            p7.Image = MC902.Properties.Resources.Green11;
            Application.DoEvents();
            ShortNode[i] = rn7;
    transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
            i++;
         rn7 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";
            System.Threading.Thread.Sleep(1000);
            for (int j = 0; j < ShortNode.Length; j++)
            {  CurrentPath += ShortNode[j].ToString();
                CurrentPath += "==#>";       }
            lblpath.Text = CurrentPath;
            send();     }
```

```
        else if (N3.Checked == true)
        {   p1.Image = MC902.Properties.Resources.Green11;
            Application.DoEvents();
            ShortNode[i] = rn1;
   transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
           i++;
  rn1 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) + "(Bits)
Transmited";
            System.Threading.Thread.Sleep(1000);
            p4.Image = MC902.Properties.Resources.Green11;
            Application.DoEvents();
            ShortNode[i] = rn4;
  transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
            i++;
         rn4 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";
            System.Threading.Thread.Sleep(1000);

            p8.Image = MC902.Properties.Resources.Green11;
            Application.DoEvents();
            ShortNode[i] = rn8;
transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
            i++;
        rn8 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";
            System.Threading.Thread.Sleep(1000);

            p12.Image = MC902.Properties.Resources.Green11;
            Application.DoEvents();
            ShortNode[i] = rn12;
  transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
            i++;
```

```
        rn12 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";

            System.Threading.Thread.Sleep(1000);

            for (int j = 0; j < ShortNode.Length; j++)

            {   CurrentPath += ShortNode[j].ToString();

                CurrentPath += "==#>";      }

            lblpath.Text = CurrentPath;

            send();      }

        else if (N4.Checked == true)

        {   p1.Image = MC902.Properties.Resources.Green11;

            Application.DoEvents();

            ShortNode[i] = rn1;

    transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";

            i++;

        rn1 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";

            System.Threading.Thread.Sleep(1000);


            p14.Image = MC902.Properties.Resources.Green11;

            Application.DoEvents();

            ShortNode[i] = rn14;

    transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";

            i++;

        rn14 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";

            System.Threading.Thread.Sleep(1000);

            for (int j = 0; j < ShortNode.Length; j++)

            {   CurrentPath += ShortNode[j].ToString();

                CurrentPath += "==#>";     }

            lblpath.Text = CurrentPath;

            send();     }

        else if (N5.Checked == true)

        {   p1.Image = MC902.Properties.Resources.Green11;
```

```
            Application.DoEvents();

            ShortNode[i] = rn1;

transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";

            i++;

        rn1 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";

            System.Threading.Thread.Sleep(1000);

            p15.Image = MC902.Properties.Resources.Green11;

            Application.DoEvents();

            ShortNode[i] = rn15;

transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";

            i++;

        rn15 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";

            System.Threading.Thread.Sleep(1000);

            for (int j = 0; j < ShortNode.Length; j++)

            {   CurrentPath += ShortNode[j].ToString();

                CurrentPath += "==#>";        }

            lblpath.Text = CurrentPath;

            send();        }

        else if (N6.Checked == true)

        {   p1.Image = MC902.Properties.Resources.Green11;

            Application.DoEvents();

            ShortNode[i] = rn1;

transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";

            i++;

          rn1 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) +
        "(Bits) Transmited";

            System.Threading.Thread.Sleep(1000);

            p11.Image = MC902.Properties.Resources.Green11;

            Application.DoEvents();

            ShortNode[i] = rn11;

transByte[i] = Convert.ToString(ReceiverCode.send.Length) + "(Bits)";
```

```
            i++;
rn11 += "\r\n " + Convert.ToString(ReceiverCode.send.Length) + "(Bits)
Transmited";
            System.Threading.Thread.Sleep(1000);
            for (int j = 0; j < ShortNode.Length; j++)
            {   CurrentPath += ShortNode[j].ToString();
                CurrentPath += "==#>";       }
            lblpath.Text = CurrentPath;
            send();    }     }
  public void send()
  {  try {
            IPAddress[] ipAddress = Dns.GetHostAddresses(txtIp.Text );
            IPEndPoint ipEnd = new IPEndPoint(ipAddress[0], 5656);
              Socket clientSock = new Socket(AddressFamily.InterNetwork,
              SocketType.Stream, ProtocolType.IP);
            clientSock.Connect(ipEnd);
            clientSock.Send(ReceiverCode.send);
            System.Threading.Thread.Sleep(1000);
            clientSock.Close();          }
       catch (Exception ex)
              {if (ex.Message == "A connection attempt failed because the
              connected party did not properly respond after a period of
              time, or established connection failed because connected host
              has failed to respond")
            { MessageBox.Show("No Such System Available Try other IP");}
            else
              {if (ex.Message == "No connection could be made because the
              target machine actively refused it")     {
                    MessageBox.Show("File Sending fail. Because Client not
                    running.");
                }   else
         { MessageBox.Show("File Sending fail." + ex.Message); }
            }  } }
  private void timer3_Tick(object sender, EventArgs e)
```

```
  {  if (ReceiverCode.Rout == "Start")
      {   water = true;
          timer3.Enabled = false;
          Routing();
          ReceiverCode.Rout = "";      } }
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
  {    obj.StartServer();    }

  private void button1_Click_1(object sender, EventArgs e)
  {   water = true;
      Routing();  }

  private void Refresh_Click(object sender, EventArgs e)
  {   timer3.Enabled = true;
      p2.Image = MC902.Properties.Resources.Orange2;
      p1.Image = MC902.Properties.Resources.Orange2;
      p3.Image = MC902.Properties.Resources.Orange2;
      p4.Image = MC902.Properties.Resources.Orange2;
      p5.Image = MC902.Properties.Resources.Orange2;
      p6.Image = MC902.Properties.Resources.Orange2;
      p7.Image = MC902.Properties.Resources.Orange2;
      p8.Image = MC902.Properties.Resources.Orange2;
      p9.Image = MC902.Properties.Resources.Orange2;
      p10.Image = MC902.Properties.Resources.Orange2;
      p11.Image = MC902.Properties.Resources.Orange2;
      p12.Image = MC902.Properties.Resources.Orange2;
      p13.Image = MC902.Properties.Resources.Orange2;
      p14.Image = MC902.Properties.Resources.Orange2;
      p15.Image = MC902.Properties.Resources.Orange2;
      seed = true;
      loadlen();
```

```
    button2.Enabled = true;     }
private void button2_Click(object sender, EventArgs e)
{   Chart c = new Chart();
    c.val = new double[2];
    c.val[0] = Convert.ToDouble(34);
    c.val[1] = Convert.ToDouble(49);
    c.txt = new string[2];
    c.txt[0] = Convert.ToString("");
    c.txt[1] = Convert.ToString("");
    c.Show();
    button2.Enabled = true;      }
private void p1_MouseEnter(object sender, EventArgs e)
{  toolTip1.SetToolTip(p1, rn1); }
private void p2_MouseEnter(object sender, EventArgs e)
{  toolTip2.SetToolTip(p2, rn2); }
private void p6_MouseEnter(object sender, EventArgs e)
{  toolTip6.SetToolTip(p6, rn6); }
private void p5_MouseEnter(object sender, EventArgs e)
{  toolTip5.SetToolTip(p5, rn5); }
private void p4_MouseEnter(object sender, EventArgs e)
{  toolTip4.SetToolTip(p4, rn4); }
private void p3_MouseEnter(object sender, EventArgs e)
{  toolTip3.SetToolTip(p3, rn3); }
private void p7_MouseEnter(object sender, EventArgs e)
{  toolTip7.SetToolTip(p7, rn7); }
private void p8_MouseEnter(object sender, EventArgs e)
{  toolTip8.SetToolTip(p8, rn8); }
private void p9_MouseEnter(object sender, EventArgs e)
{  toolTip9.SetToolTip(p9, rn9); }
private void p10_MouseEnter(object sender, EventArgs e)
{  toolTip10.SetToolTip(p10, rn10); }
```

```
    private void p11_MouseEnter(object sender, EventArgs e)
    {  toolTip11.SetToolTip(p11, rn11); }
    private void p12_MouseEnter(object sender, EventArgs e)
    {  toolTip12.SetToolTip(p12, rn12); }
    private void p13_MouseEnter(object sender, EventArgs e)
    {  toolTip13.SetToolTip(p13, rn13); }
    private void p14_MouseEnter(object sender, EventArgs e)
    {  toolTip14.SetToolTip(p14, rn14); }
    private void p15_MouseEnter(object sender, EventArgs e)
    {  toolTip15.SetToolTip(p15, rn15); }
    private void N1_CheckedChanged(object sender, EventArgs e)
    {   ShortNode = new string[2];
        transByte =new string[2];   }
    private void N2_CheckedChanged(object sender, EventArgs e)
    {   ShortNode = new string[3];
        transByte = new string[3];  }
    private void N3_CheckedChanged(object sender, EventArgs e)
    {   transByte = new string[4];
        ShortNode = new string[4];   }
    private void N4_CheckedChanged(object sender, EventArgs e)
    {   ShortNode = new string[2];
        transByte = new string[2];   }
    private void N5_CheckedChanged(object sender, EventArgs e)
    {   ShortNode = new string[2];
        transByte = new string[2];    }
    private void N6_CheckedChanged(object sender, EventArgs e)
    {   ShortNode = new string[2];
        transByte = new string[2];   }    }
class ReceiverCode
{   IPEndPoint ipEnd;
    Socket sock;
```

```
        public ReceiverCode()
        {   IPHostEntry ipEntry = Dns.GetHostEntry(Environment.MachineName);
            IPAddress IpAddr = ipEntry.AddressList[0];
            ipEnd = new IPEndPoint(IpAddr, 5655);
            sock = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
            ProtocolType.IP);
            sock.Bind(ipEnd); }
        public static string receivedPath;
        public static string curMsg = "Stopped";
        public static string Rout = "";
        public static string FileSize = "";
        public static string MsgStatus = "";
        public static byte[] send;
        public void StartServer()
        {  try  {   curMsg = "Starting...";
                sock.Listen(100);
                curMsg = "Running and waiting to receive file.";
                Socket clientSock = sock.Accept();
                byte[] clientData = new byte[1024 * 5000];
                int receivedBytesLen = clientSock.Receive(clientData);
                curMsg = "Receiving data...";
                send = new byte[receivedBytesLen];
                Array.Copy(clientData, send, receivedBytesLen);
                Rout = "Start";
                clientSock.Close();
                curMsg = "Reeived & Saved file; Server Stopped.";
                StartServer();      }
            catch (Exception ex)  {  curMsg = "File Receving error.";
            }
        }
    }
}
```

9. TESTING

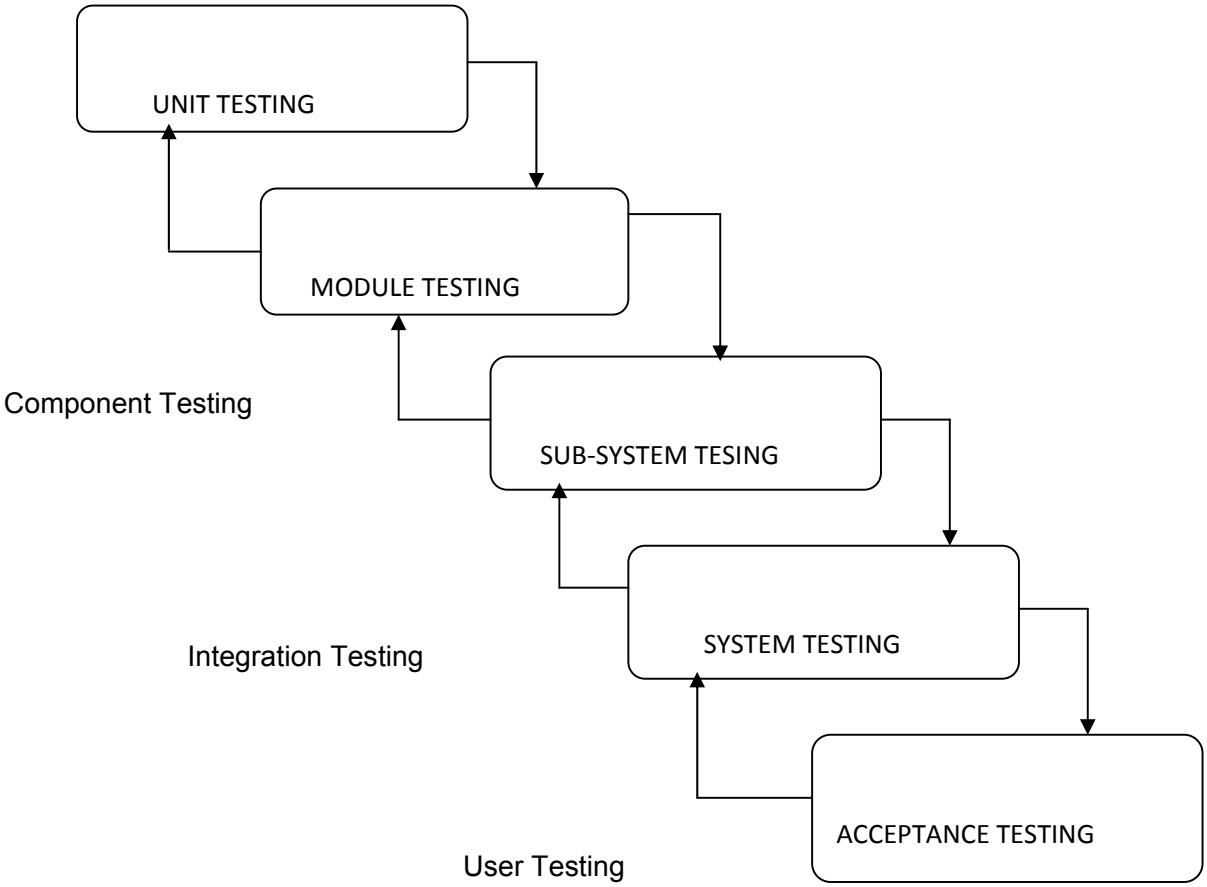
9.1 Introduction

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software we spiral in along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally we arrive at system testing, where the software and other system elements are tested as a whole.



9.2 Unit Testing

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing we have is white box oriented and some modules the steps are conducted in parallel.

1. White Box Testing

This type of testing ensures that

- All independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity.

To follow the concept of white box testing we have tested each form .we have created independently to verify that Data flow is correct, All conditions are exercised to check their validity, All loops are executed on their boundaries.

2. Basic Path Testing

Established technique of flow graph with Cyclomatic complexity was used to derive test cases for all the functions. The main steps in deriving test cases were:

Use the design of the code and draw correspondent flow graph.

Determine the Cyclomatic complexity of resultant flow graph, using formula:

V(G)=E-N+2 or

V(G)=P+1 or

V(G)=Number Of Regions

Where V(G) is Cyclomatic complexity,

E is the number of edges,

N is the number of flow graph nodes,

P is the number of predicate nodes.

Determine the basis of set of linearly independent paths.

3. Conditional Testing

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested. So that each path that may be generate on particular condition is traced to uncover any possible errors.

4. Data Flow Testing

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variable were declared. The *definition-use chain* method was used in this type of testing. These were particularly useful in nested statements.

5. Loop Testing

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

All the loops were tested at their limits, just above them and just below them.

All the loops were skipped at least once.

For nested loops test the inner most loop first and then work outwards.

For concatenated loops the values of dependent loops were set with the help of connected loop.

Unstructured loops were resolved into nested loops or concatenated loops and tested as above.

Each unit has been separately tested by the development team itself and all the input have been validated.

9.3 Integration Testing

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works.

The purpose of integration testing is to verify functional, performance and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface.

Test cases are constructed to test that all components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing.

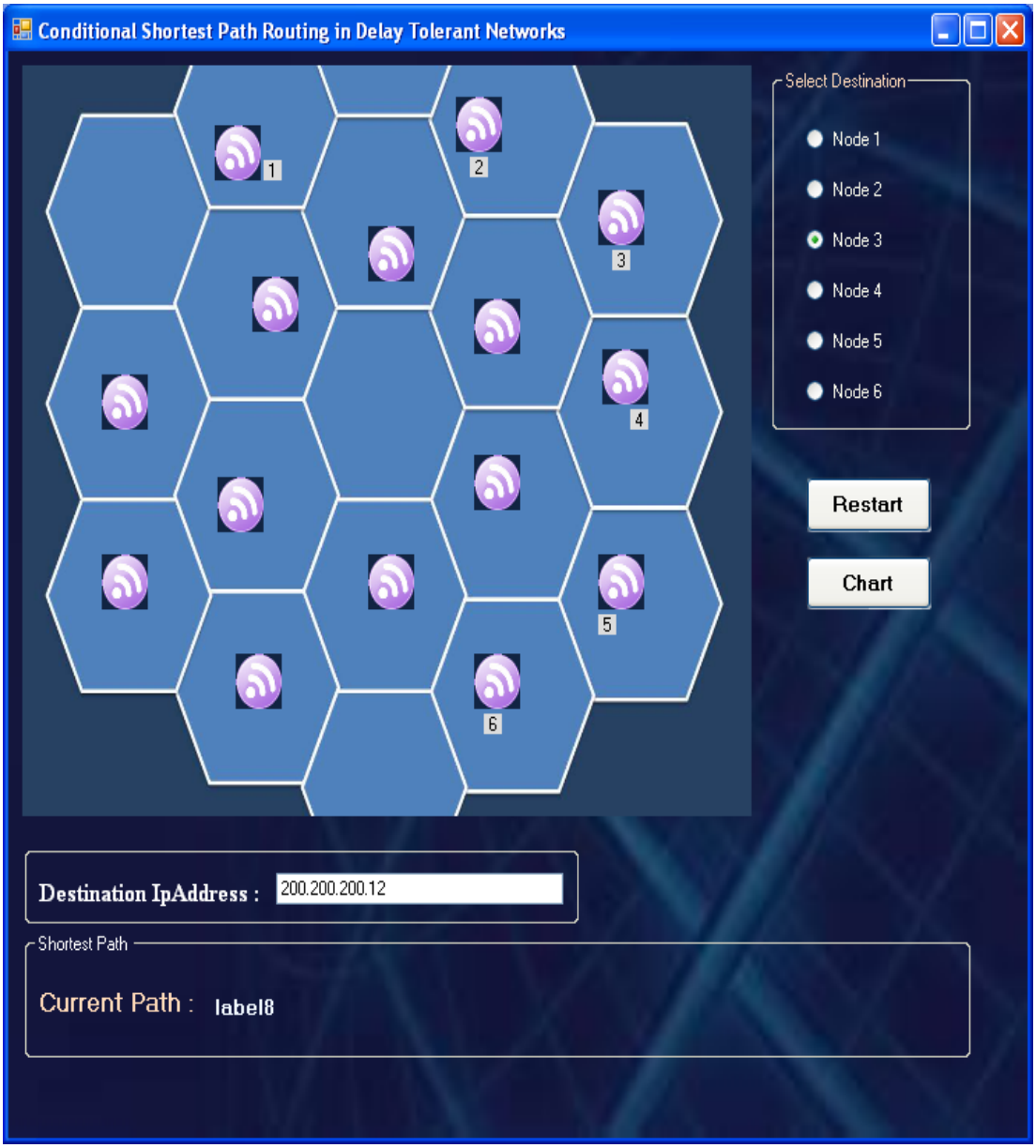
The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

10. OUTPUT SCREENS

Selecting the Receiving Path by Client:

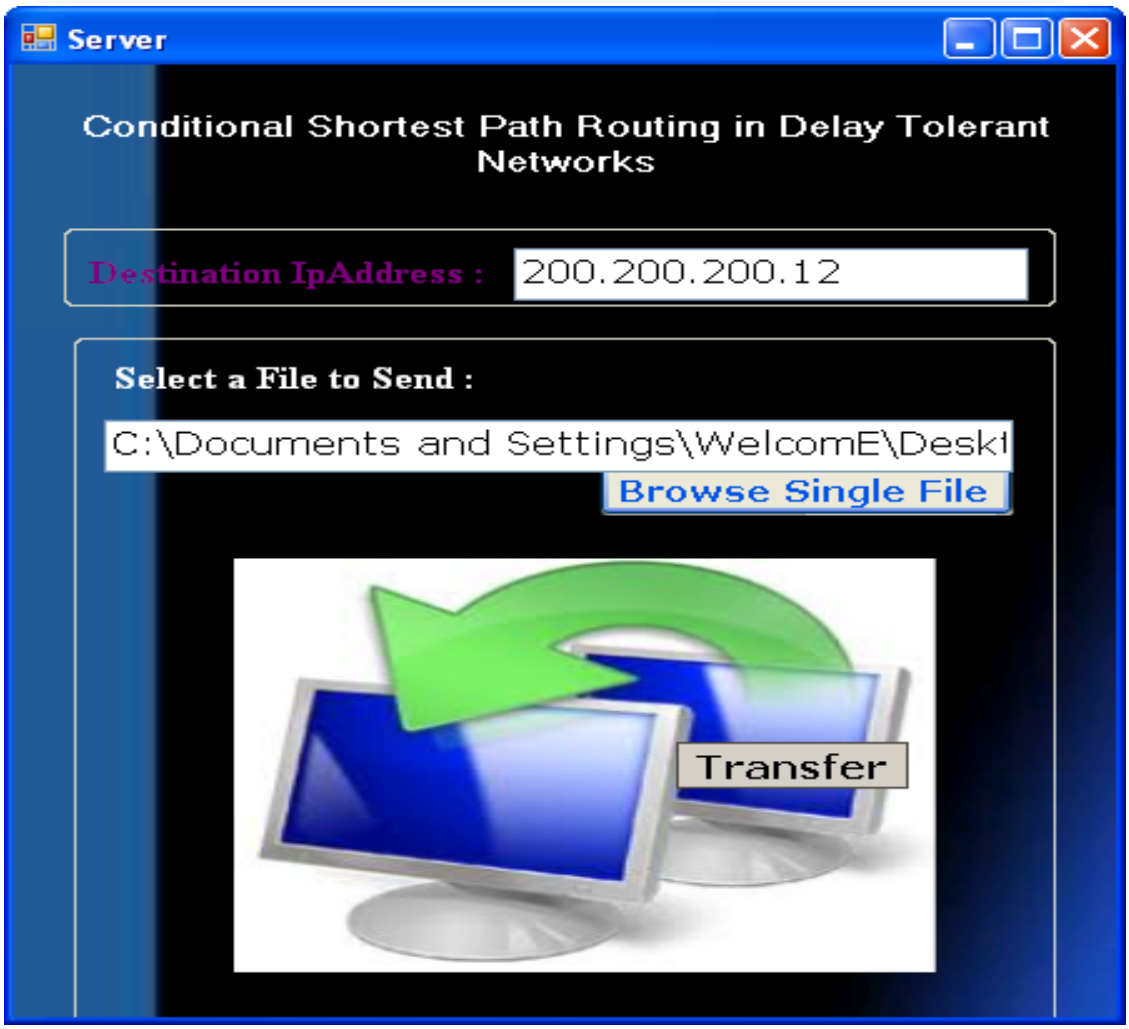
Scr-1. Selecting Receiving Path at Client Side

Selecting Destination node and IP Address in Router:



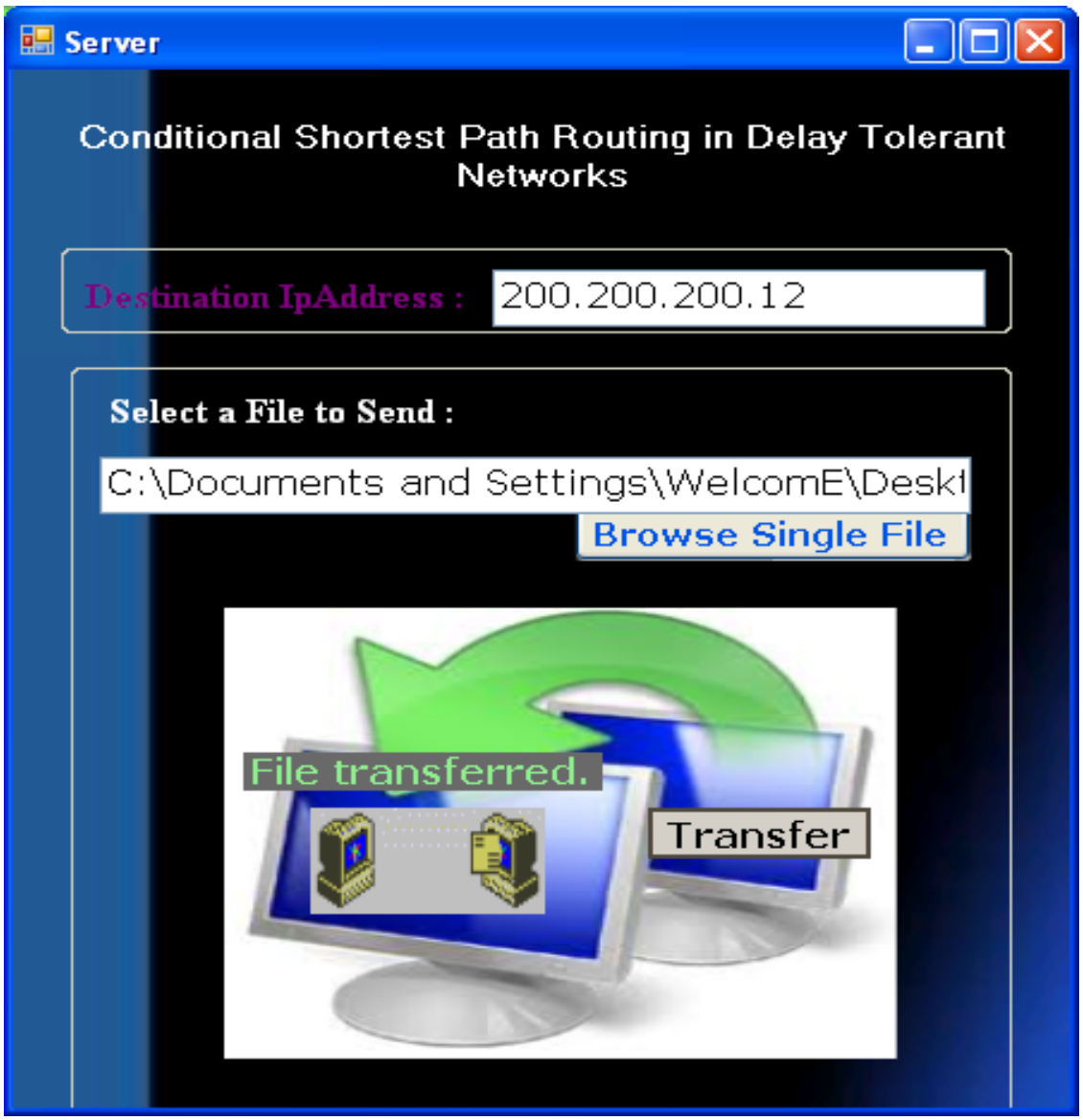
Scr-2. Select Destination Node and IP Address in Router

Enter Destination IP Address and Select a File to Send in Server:



Scr-3. Choosing Destination IP Address and Select a File to Send in Server

File Transferring from server to client:



Scr-4. File Transferring from server to client

Conditional Shortest Path Routing in Delay Tolerant N/w:

Scr-5. Conditional Shortest Path Routing in Delay Tolerant N/w

File Received by client:

Scr-6. File Received by client:

12. CONCLUSION

In this paper, we introduced a new metric called conditional intermeeting time inspired by the results of the recent studies showing that nodes' intermeeting times are not memoryless and that motion patterns of mobile nodes are frequently repetitive. Then, we looked at the effects of this metric on shortest path based routing in DTN's. For this purpose, we updated the shortest path based routing algorithms using conditional intermeeting times and proposed to route the messages over conditional shortest paths. Finally, we ran simulations to evaluate the proposed algorithm and demonstrated the superiority of CSPR protocol over the existing shortest path routing algorithms. In future work, we will look at the performance of the proposed algorithm in different data sets to see the effect of conditional intermeeting time in different environments. Moreover, we will consider extending our CSPR algorithm by using more information (more than one known meetings) from the contact history while deciding conditional intermeeting times. For this, we plan to use probabilistic context free grammars (PCFG) and utilize the construction algorithm presented in . Such a model will be able to hold history information concisely, and provide further generalizations for unseen data.

13. REFERENCES

1. *Delay tolerant networking research group*, http://www.dtnrg.org.

2. Y. Wang, S. Jain, M. Martonosi, and K. Fall, Erasure coding based routing for opportunistic networks, in *Proceedings of ACM SIGCOMM Workshop on Delay Tolerant Networking (WDTN), 2005*.

3. S. Jain, K. Fall, and R. Patra, Routing in a delay tolerant network, in *Proceedings of ACM SIGCOMM, Aug. 2004.*

4. E. P. C. Jones, L. Li, and P. A. S. Ward, Practical routing in delay tolerant networks, in *Proceedings of ACM SIGCOMM workshop on Delay Tolerant Networking (WDTN), 2005*.

5. C. Liu and J. Wu, An Optimal Probabilistically Forwarding Protocol in *Delay Tolerant Networks, in Proceedings of MobiHoc, 2009.*

6. E. Bulut, Z. Wang, and B. Szymanski, Cost-Effective Multi-Period Spraying for Routing in Delay Tolerant Networks, to appear in *IEEE/ACM Transactions on Networking, 2010.*

7. K. Fall. A delay–tolerant network architecture for challenged internets. In *ACM SIGCOMM, 2003.*

8. T. Spyropoulos, K. Psounis,C. S. Raghavendra, Efficient routing in intermittently connected mobile networks: The single-copy case, *IEEE/ACM Transactions on Networking, vol. 16, no. 1, 2008.*

9. J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, MaxProp: Routing for Vehicle-Based Disruption- Tolerant Networks, *In Proc. IEEE Infocom, 2006.*

10. A. Vahdat and D. Becker, Epidemic routing for partially connected ad hoc networks, *Duke University, Tech. Rep. CS-200006, 2000.*

11. T. Spyropoulos, K. Psounis,C. S. Raghavendra, Efficient routing in intermittently connected Mobile networks: The multi-copy case, *IEEE/ACM Transactions on Networking, 2008*.

12. T. Spyropoulos, K. Psounis, C. S. Raghavendra, Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks, *ACM SIGCOMM Workshop, 2005*.

13. A. Lindgren, A. Doria, and O. Schelen, Probabilistic routing in intermittently connected networks, *SIGMOBILE Mobile Computing and Communication Review, vol. 7, no. 3, 2003*.

14. A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, Impact of Human Mobility on the Design of Opportunistic Forwarding Algorithms, in *Proceedings of INFOCOM, 2006.*

15. P. U. Tournoux, J. Leguay, F. Benbadis, V. Conan, M. Amorim, J. Whitbeck, The Accordion Phenomenon: Analysis, Characterization, and Impact on DTN Routing, in *Proceedings of Infocom, 2009.*

16. C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM, 1994.*

17. C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *IEEE WMCSA, 1999.*

18. E. M. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Personal Communications, 6:46–55, 1999*

19. D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi- hop wireless routing. In *ACM MobiCom, 2003.*

17. R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *ACM MobiCom, 2004.*

20. X. Zhang, J. F. Kurose, B. Levine, D. Towsley, and H. Zhang, Study of a Bus-Based Disruption Tolerant Network: Mobility Modeling and Impact on Routing, In *Proceedings of ACM MobiCom, 2007.*

21. S. Srinivasa and S. Krishnamurthy, CREST: An Opportunistic Forwarding Protocol Based on Conditional Residual Time, in *Proceedings of IEEE SECON, 2009.*

22. P. U. Tournoux, J. Leguay, F. Benbadis, V. Conan, M. Amorim, J. Whitbeck, The Accordion Phenomenon: Analysis, Characterization, and Impact on DTN Routing, in *Proceedings of Infocom, 2009.*

23. C. Liu and J. Wu, Routing in a Cyclic Mobispace, In Proceedings of ACM Mobihoc, 2008.

A/c Dy No: 1703
Date: 03/03/17

UNIVERSITY GRANTS COMMISSION-SOUTH EASTERN REGIONAL OFFICE
5-9-194, CHIRAG ALI LANE, IV FLOOR A.P.S.F.C. BUILDING, HYDERABAD -500 001
Phones: 040 - 23204735, 23200208 FAX: 040 – 23204734, Website: www.ugc.ac.in,ugcsero@gmail.com

No: F. MRP-5681/15 GEN/(UGC-SERO)

February, 2017

The Accounts Officer
South Eastern Regional Office
University Grants Commission
Hyderabad – 500 001

LINKNO:5681. DEPT:COMPUTER SCIENCE
COMCODE: APOS097

03 MAR 2017

Sub: Release of Grants-in-aid to **The Principal TARA GOVT. COLLEGE SANGAREDDY MEDAK DIST.502001.** Under the Scheme "Minor Research Projects" - Reg.

Sir/Madam,

On the basis of the accounts received for the grants released earlier under the scheme, I am to convey the sanction of the Commission for the payment of **Rs.10416.** to **The Principal, TARA GOVT. COLLEGE SANGAREDDY MEDAK DIST. 502001.** as final instalment towards the Minor Research Project entitled **CONDITIONAL SHORTEST PATH ROUTING IN DEL T NETWORKS** submitted by **DR. RAJITHA DEVI.G Department of COMPUTER SCIENCE** as per the details given below:-

Item	Allocation (Rs.)	Amount already released (Rs.)	Amount sanctioned now (Rs.)	Total grant sanctioned/released so far (Rs.)
Hiring Services	0 0	0 0	0 0	0 0
Contingency	20000.	10000.	10000.	20000.
Chemicals	0 0	0 0	0 0	0 0
Travel/Field Work	10000.	5000.	416.	5416.
Total	**30000.**	**15000.**	**10416.**	**25416.**
Equipment	20000.	20000.	0 0	20000.
Books	10000.	10000.	0 0	10000.
Total	**30000.**	**30000.**	**0 0**	**30000.**
Grand Total	**60000.**	**45000.**	**10416.**	**55416.**

1. The grant is debitable to following head of account.

Amount Sanctioned	Head Of Accounts	Category
Rs.10416.	31-GIA-MRP(50)-3(A)2202.03.102.02.01	GEN

2. The sanctioned amount is debitable to the **Head of Account 31-GIA-MRP(50)-3(A)-2202.03.102.02.01 (General)** and is valid for payment during the financial year 2016-17 Only and the amount of the Grant shall be drawn by the Accounts Officer (Drawing and Disbursing Officer) UGC-SERO, Hyd. on the Grants-In Aid Bill and shall be disbursed to and credited to **"The Principal, TARA GOVT. COLLEGE, SANGAREDDY, MEDAK DIST. by Electronic Mode through PFMS Portal at the following details:** ***"(a)Name & Address of Account Holder: The Principal, TARA GOVT. COLLEGE, SANGAREDDY, MEDAK DIST. (b) Account No: 62061242507(c) Name & Address of Bank Branch: STATE BANK OF HYDERABAD, SANGREDDY (d)IFSC Code:SBHY0020107 UniqueID: TLME00001515***
3. The Grant is Subject to the adjust on the basis of Utilization Certificate in the prescribed Proforma submitted by the Institution.
4. The Institution shall maintain proper accounts of the expenditure out of the Grants which shall be utilized only on the approved items of expenditure.
5. The institution may follow the **General Financial Rules, 2005** and take urgent necessary action to amend their manuals of financial procedures to bring them in conformity with GFRs, 2005 and those don't have their own approved manuals on financial procedures may adopt the provision of GFRs 2005 and instructions / Guidelines there under from time to time.
6. The Utilization Certificate to the effect that the grant has been utilized for the propose for which it has been sanctioned shall be furnished to UGC as early as possible after the close of current financial year.
7. The assets acquired wholly or substantially out of UGC's Gant shall NOT be disposed or encumbered or utilized for the proposes other than those for which the grants was given without proper sanction of the UGC and should at any time the Institution ceased to function, such assets shall revert to the University Grants Commission.
8. A Register of Assets acquired wholly or substantially out of the Grant shall be maintained by the Institution in the prescribed proforma.
9. The Grantee Institution shall ensure the utilization of Grants-in-Aid for which it is being sanctioned / paid. In case of Non-Utilization / Part Utilization thereof, simple interest **@ 10% per annum**, as amended from time to time on the unutilized amount from the date of credit of amount to the date of refund as per provision contained in General Financial Rules of Govt. of India will be charged.
10. The Institution shall follow strictly the Government of India/ UGC's Guidelines regarding implementation of the reservation policy (Both Vertical [for SC,ST & OBC] and horizontal (for Personas with disability etc.]) in teaching and non-teaching posts.

P.T.O.